



A Cost-Effective Near-Storage Processing Solution for Offline Inference of Long-Context LLMs

Seoul National University

Jinho Lee

Profile

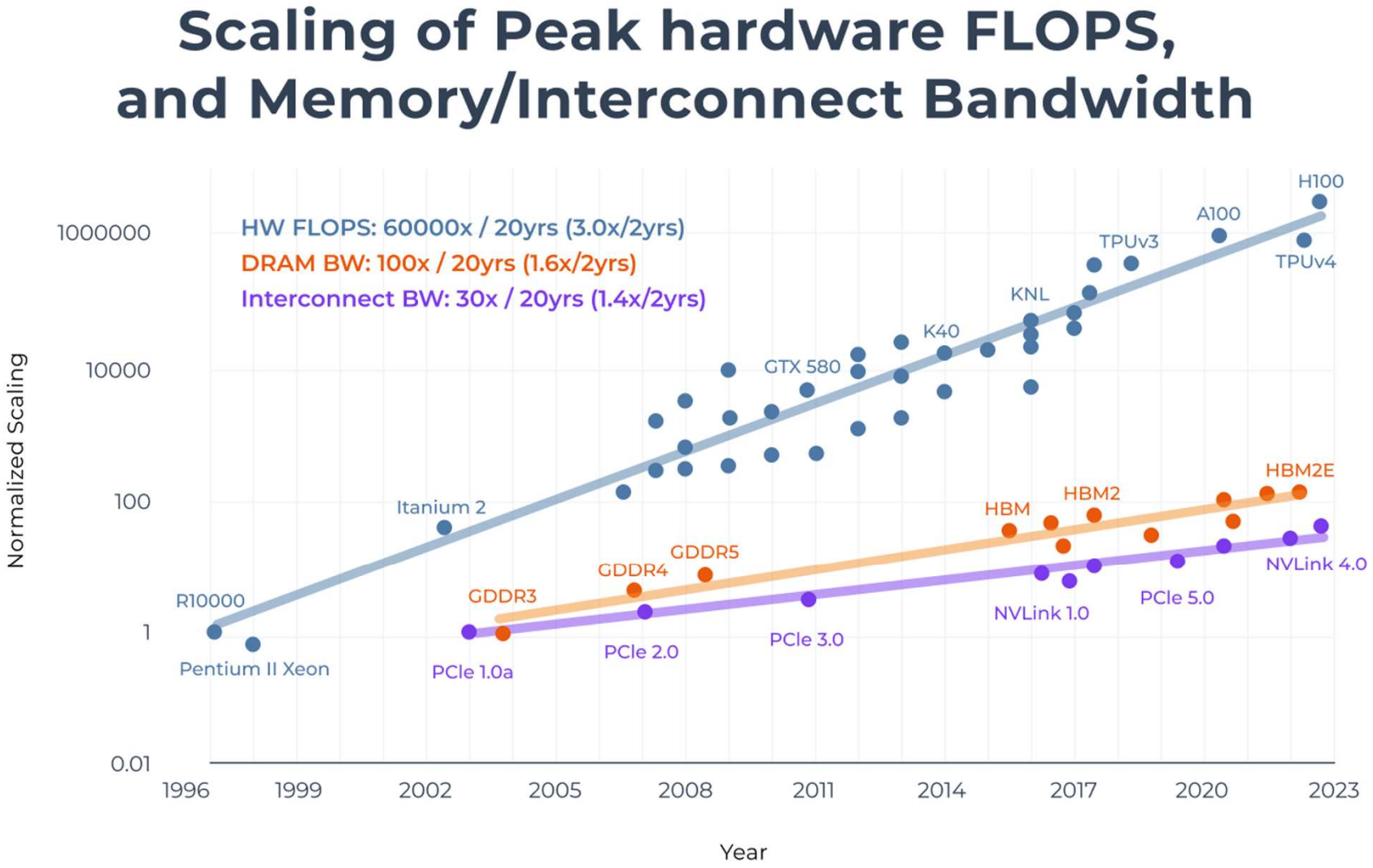
- Jinho Lee
- ECE, SNU 2022.9 – current
- CS/AI, Yonsei Univ. 2019.9 – 2022.08
- IBM, Austin, TX 2015.8 – 2019.08
- Ph.D. ECE, SNU 2005.3 – 2016.02

  Advisor: Prof. Kiyoung Choi

- Research Interests
  - Computer Architecture and Systems (for AI)

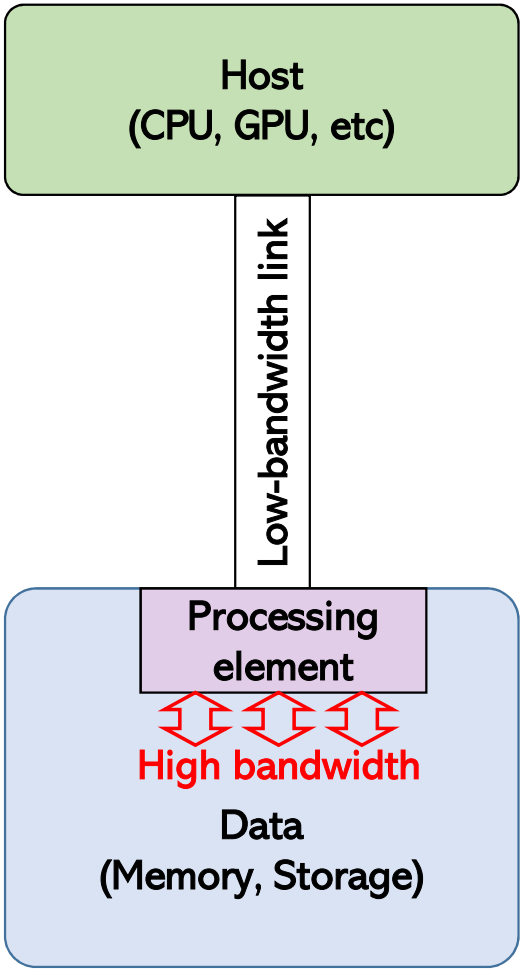


The Memory Wall



Near-Data Processing

- A promising solution to the bandwidth bottleneck.
- Achieves higher bandwidth by offloading computation to memory.
- DRAM, SRAM, Flash, CXL, etc



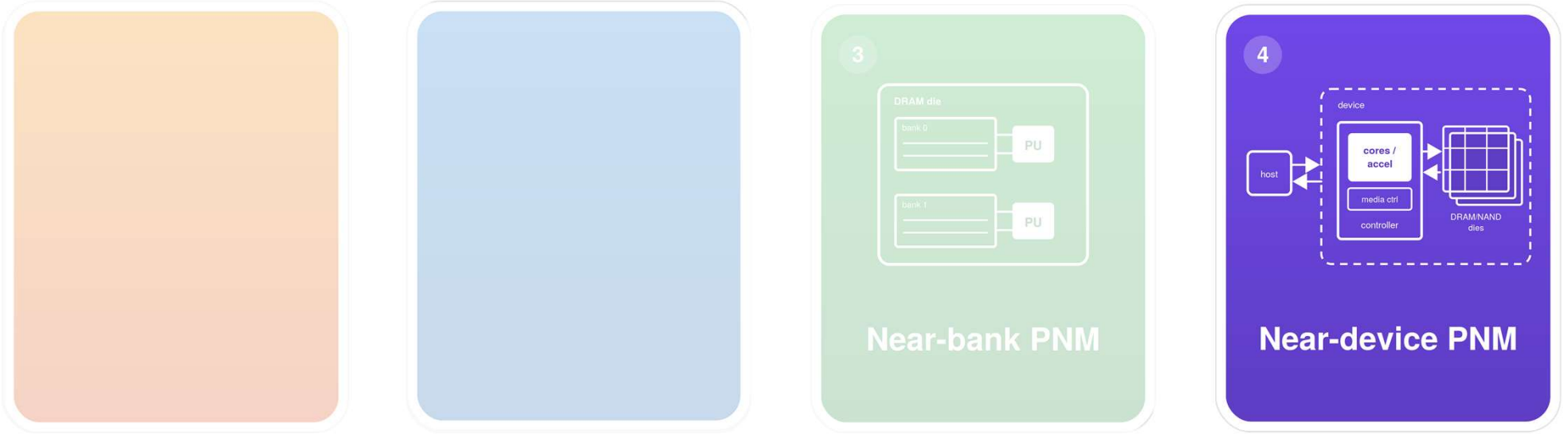
Near-Data Processing



- **PUM** (Processing-Using-Memory): Using existing circuitry of memory for computation
- RRAM/SRAM Crossbar, DRAM triple row activation, etc

- **PNM** (Processing Near Memory): Putting extra logic near data array for internal bandwidth
- DRAM Bank, DRAM Rank, Flash page, SSD controller, etc

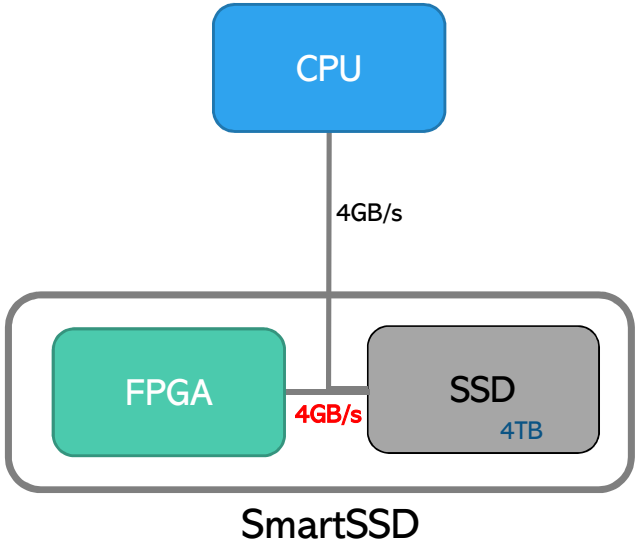
Near-Data Processing

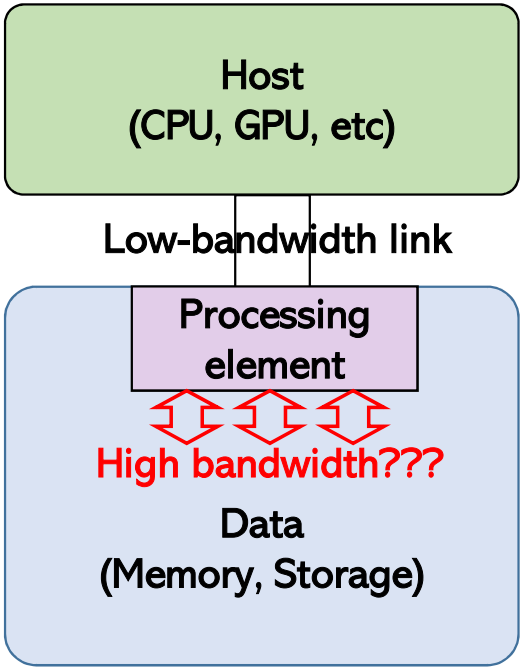


- **PNM** (Processing Near Memory): Putting extra logic near data array for internal bandwidth
- SSD

SmartSSD (2021) Features



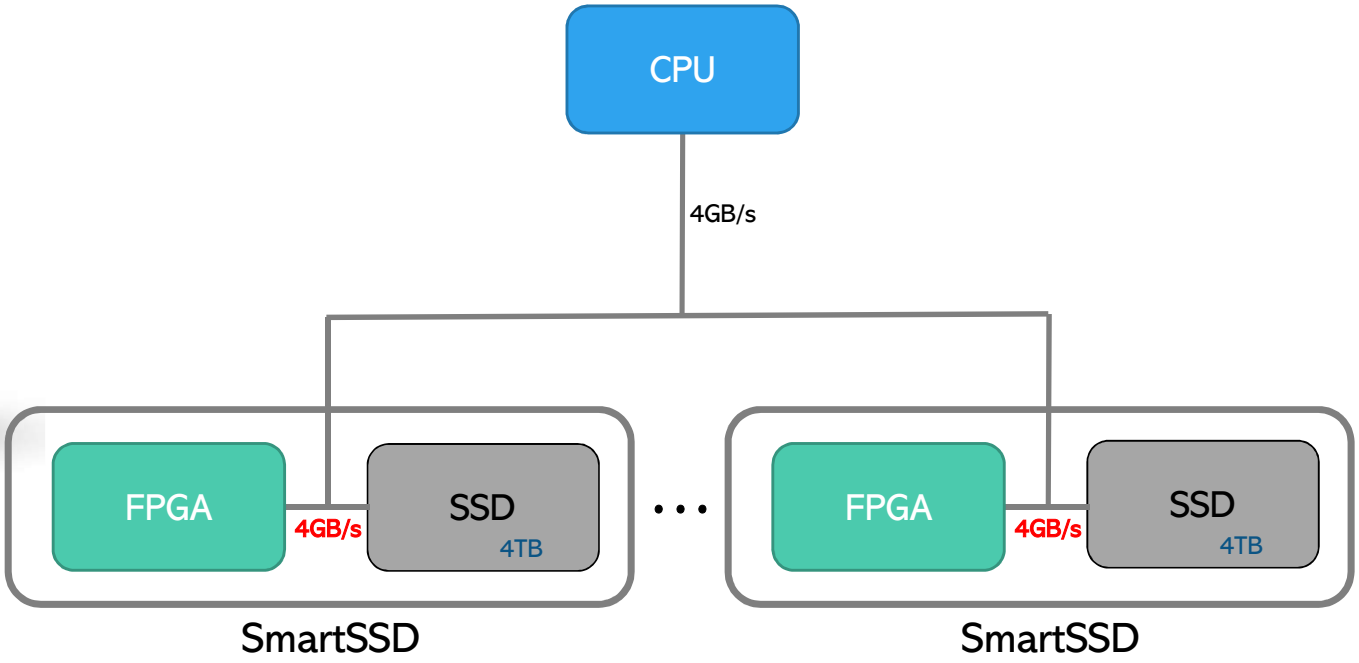




- FPGA inside SSD case
  - Xilinx Kintex Ultrascale+ KU15p
    - A small FPGA
    - SSD Power budget 25W
  - Internal 4GB DDR4 DRAM
- Same 4GB/s for FPGA←→SSD
  - Higher internal bandwidth for many SmartSSD's (say, 16x)

SmartSSD Features





- FPGA inside SSD case
  - Xilinx Kintex Ultrascale+ KU15p
    - A small FPGA
    - SSD Power budget 25W
  - Internal 4GB DDR4 DRAM
- Same 4GB/s for FPGA←→SSD
  - Higher internal bandwidth for many SmartSSD's (say, 16x)

SmartSSD Features



- FPGA inside SSD case
  - Xilinx Kintex Ultrascale+ KU15p
    - A small FPGA
    - SSD Power budget 25W
  - Internal 4GB DDR4 DRAM
- Same 4GB/s for FPGA←→SSD
  - Higher internal bandwidth for many SmartSSD's (say, 16x)

- FPGA gives you all the **freedom** for general applications
- But FPGAs are **difficult** to use
  - Verilog is difficult
  - Xilinx tools (Vivado) are inconvenient…



SmartSSD Features



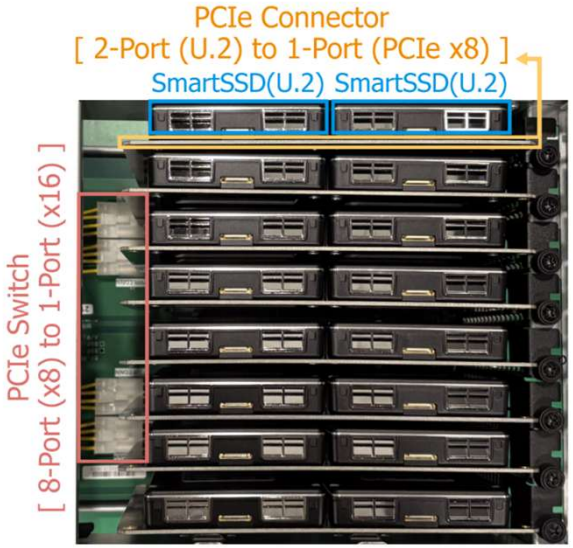
- FPGA inside SSD case
  - Xilinx Kintex Ultrascale+ KU15p
    - A small FPGA
    - SSD Power budget 25W
  - Internal 4GB DDR4 DRAM
- Same 4GB/s for FPGA←→SSD
  - Higher internal bandwidth for many SmartSSD's (say, 16x)

- This is not strong enough
- Only 25W - Limited by the SSD power budget
- The DDR4 only provides ~20GB/s
  - 4GB is too small
- FPGA is expensive in small production
  - ~$3,000
  - Much more than a typical SSD (~$200)
  - Maybe relieved with mass production

SmartSSD Features







- FPGA inside SSD case
  - Xilinx Kintex Ultrascale+ KU15p
    - A small FPGA
    - SSD Power budget 25W
  - Internal 4GB DDR4 DRAM
- Same 4GB/s for FPGA←→SSD
  - Higher internal bandwidth for many SmartSSD's (say, 16x)

- We need 8x~16x of these for meaningful speedup
  - $3,000 * 16 = $48,000
- This often requires an expander
  - ~$10,000 additional cost

SmartSSD – What do we do?

- LLMs
  - What else?



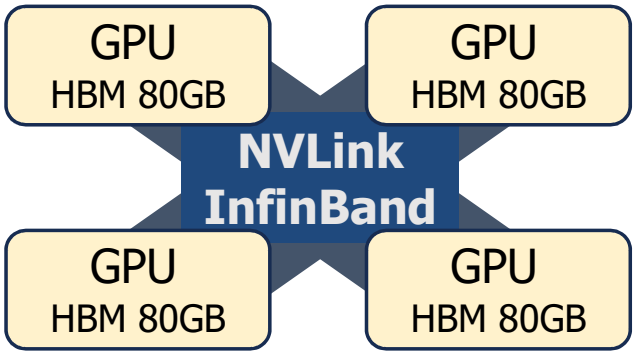
- LLM Training
  - H. Jang+, "Smart-Infinity", HPCA 2024
    - Best paper honorable mention
  - Target gradient descent
  - Put optimizer states to SSD
  - Apply gradient compression for more speedup

- LLM Inference
  - H. Jang+, "HILOS", ASPLOS 2026
  - Target multi-head attention
  - Put KV cache to SSD
  - System-level optimizations for more speedup

Background: Scenario for LLM systems

How to handle the massive increase in required data size?

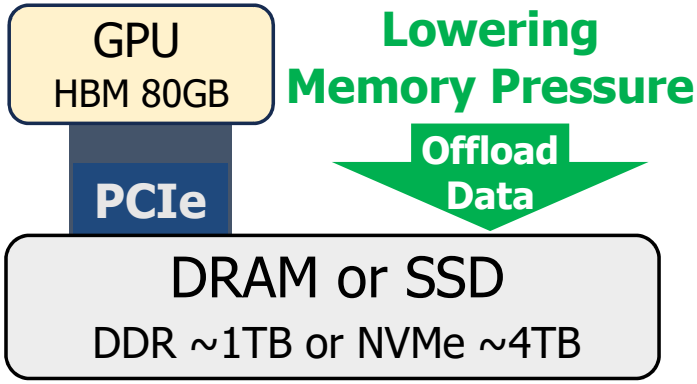
1. GPU-only Solutions
   - **Sharding data across multiple GPUs**
   - **(+) Low latency** for real-time interaction
   - **(–) High cost**: Requires expensive HBM and NVLink
   - **Example: Pretraining, Interactive chatbots**



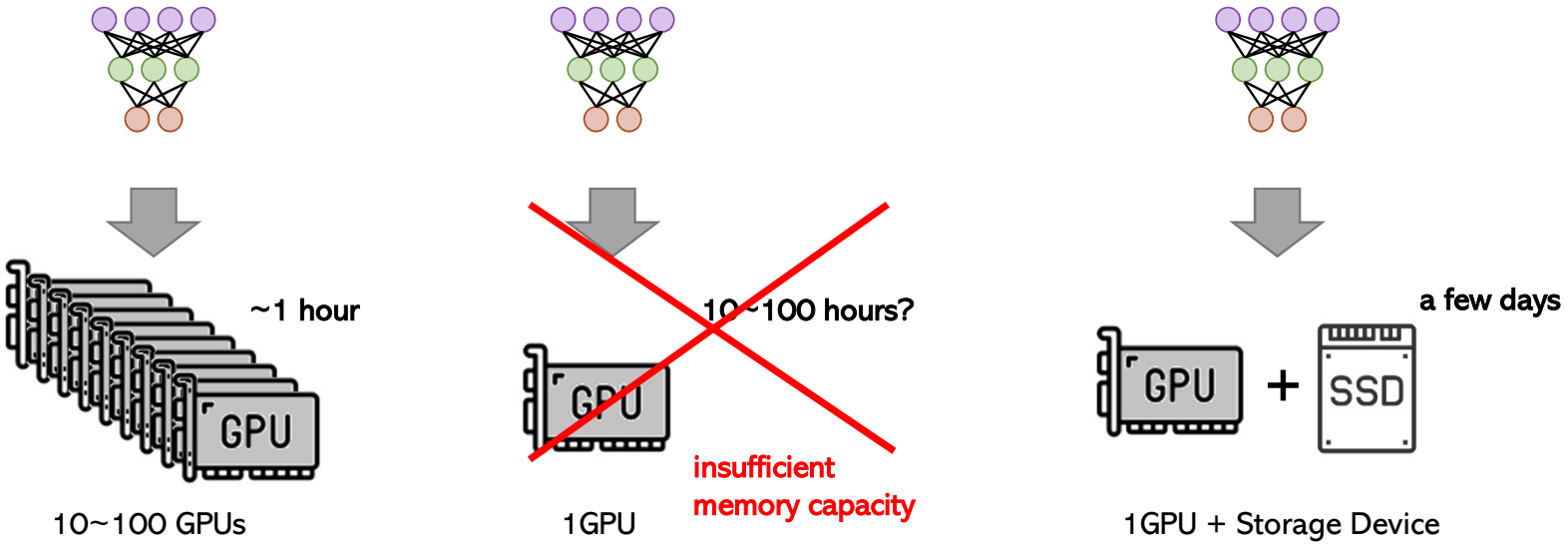
2. Storage-Offloaded

Our research scope

   - **Extends GPU memory hierarchy via DRAM/SSD**
   - **(+)** Moderate **throughput** with **cost-efficient** DRAM/SSD!
   - **(–) High latency** due to data movement bottlenecks
   - **Example: Fine-tuning, benchmarking**



Storage-Offloaded LLM ("AI Democratization")



- A 100-GPU cluster is too expensive
- Cannot even begin training/inference with a single GPU
- A storage device can be a cheap alternative

SmartSSD – What do we do?



- LLM Training
  - H. Jang+, "Smart-Infinity", HPCA 2024
    - Best paper honorable mention
  - Target gradient descent
  - Put optimizer states to SSD
  - Apply gradient compression for more speedup

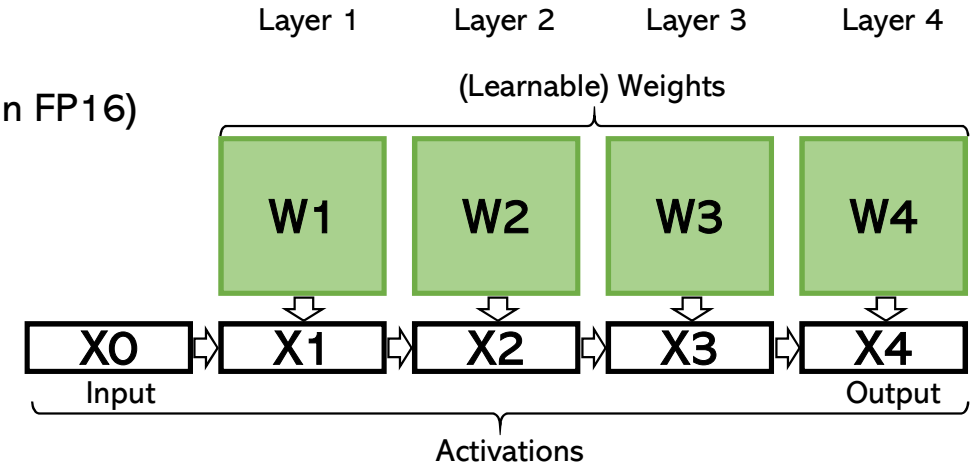
- LLM Inference
  - H. Jang+, "HILOS", ASPLOS 2026
  - Target multi-head attention
  - Put KV cache to SSD
  - System-level optimizations for more speedup

Memory Capacity for LLM Training

An example of a large language model with four layers

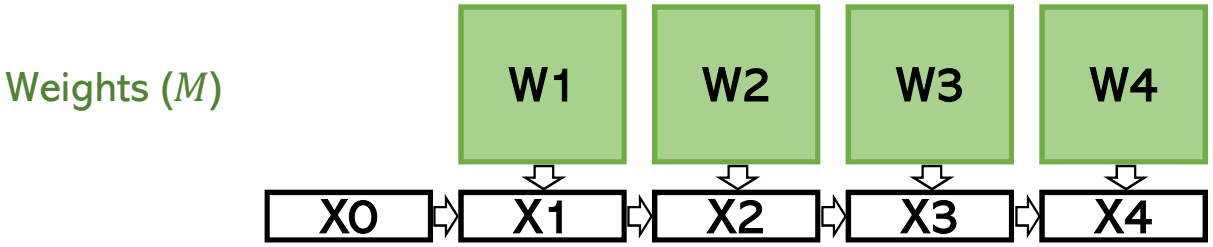
(M : Model size in FP16)

Weights (M)



Memory Capacity for LLM Training

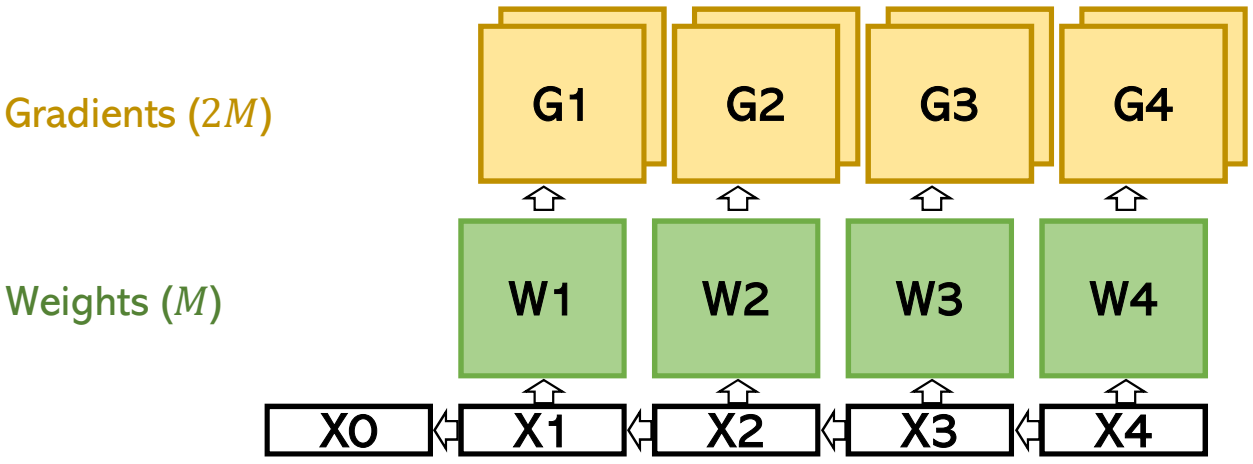
Multiply them backward to obtain gradients





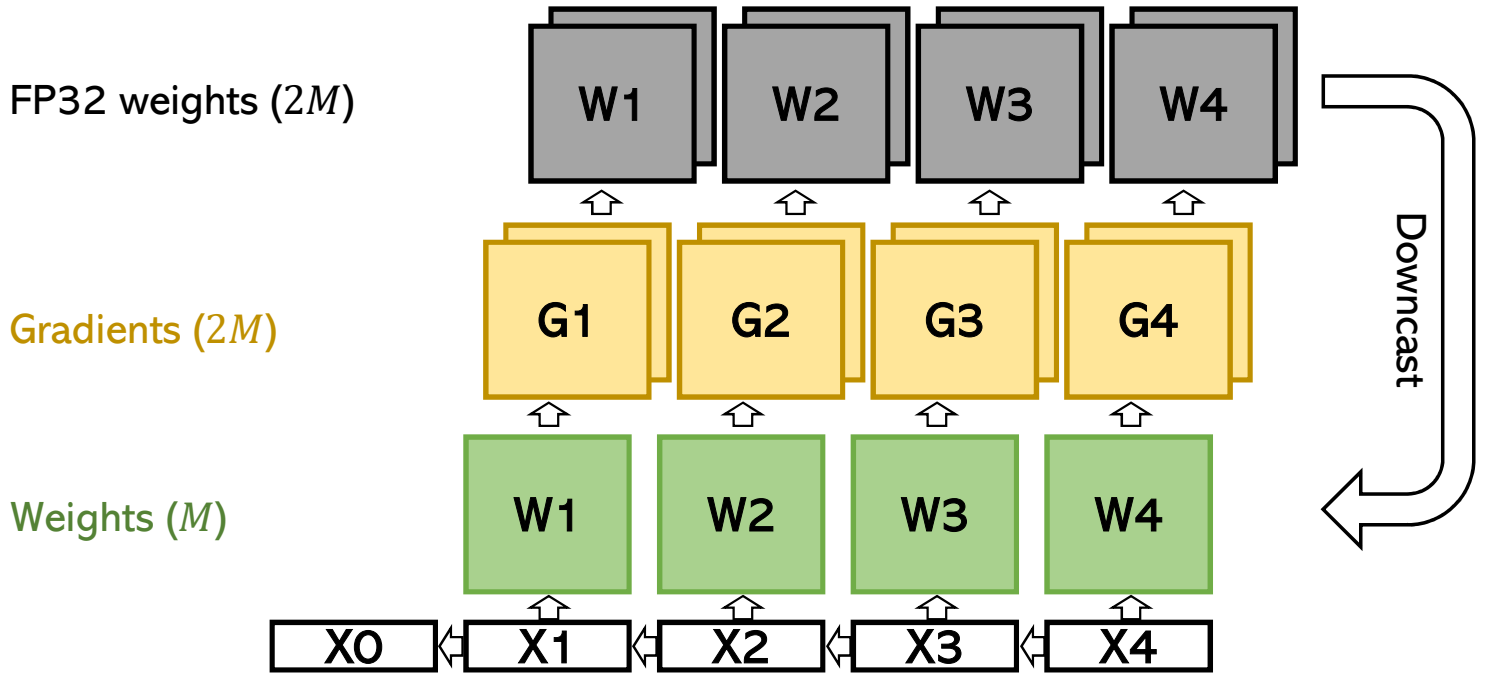
Memory Capacity for LLM Training

Multiply them backward to obtain gradients



Memory Capacity for LLM Training

Gradients are used to update next weights $\overrightarrow{\theta_{t+1}} = \overrightarrow{\theta_t} + \alpha \cdot \overrightarrow{g_t}$



Memory Capacity for LLM Training

$$\overrightarrow{\theta_{t+1}} = \overrightarrow{\theta_t} + \alpha \cdot \overrightarrow{g_t} + \beta \cdot \overrightarrow{m_t} + \gamma \cdot \overrightarrow{v_t}$$

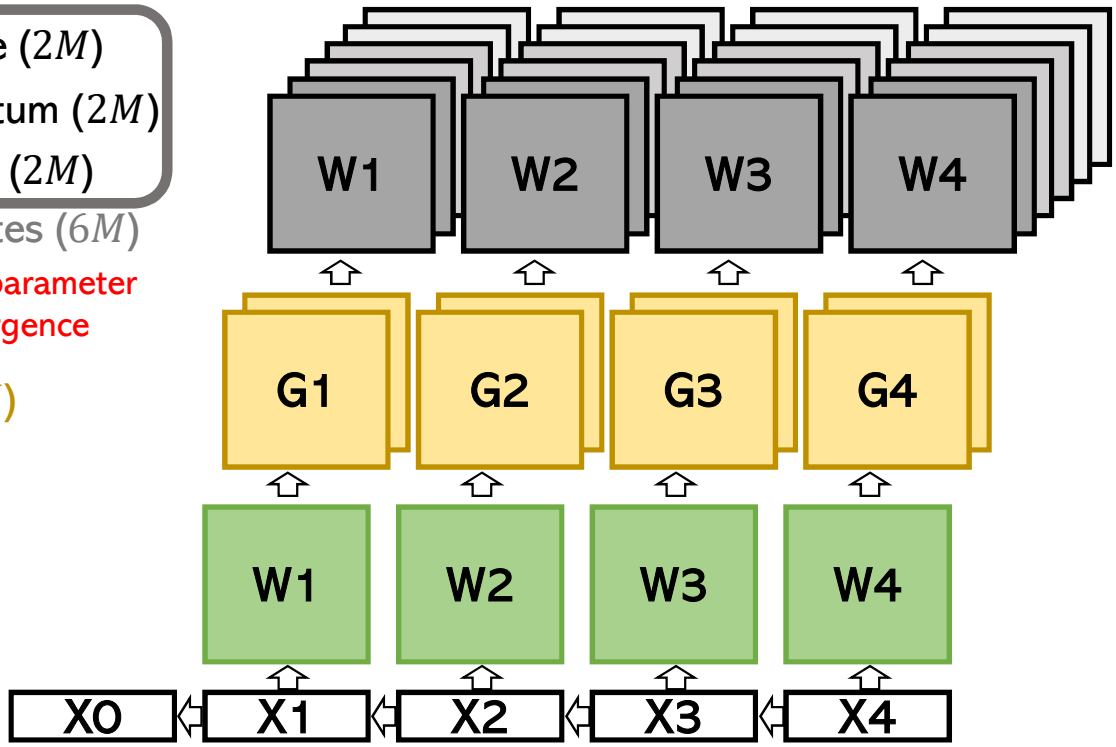
FP32 variance ($2M$)
FP32 Momentum ($2M$)
FP32 weights ($2M$)

Optimizer states ($6M$)

Maintained per parameter for better convergence

Gradients ($2M$)

Weights (M)





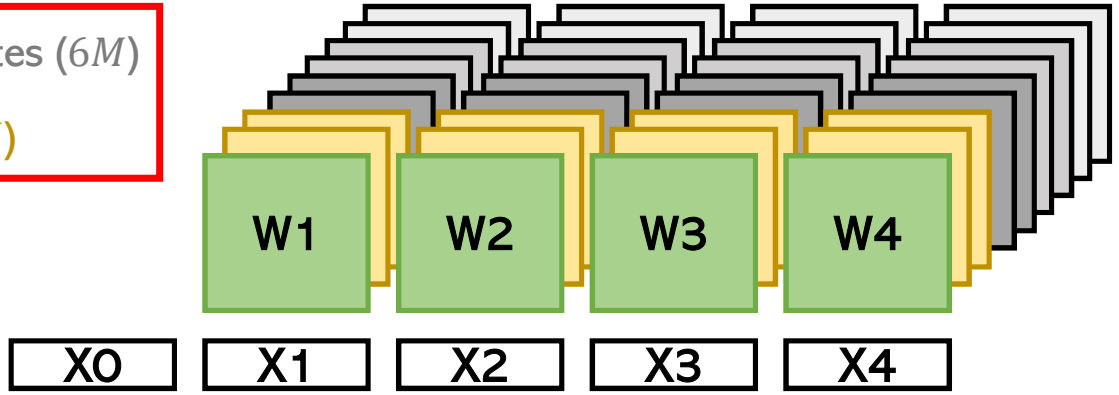
Memory Capacity for LLM Training

What consumes the most memory

Optimizer states ($6M$)

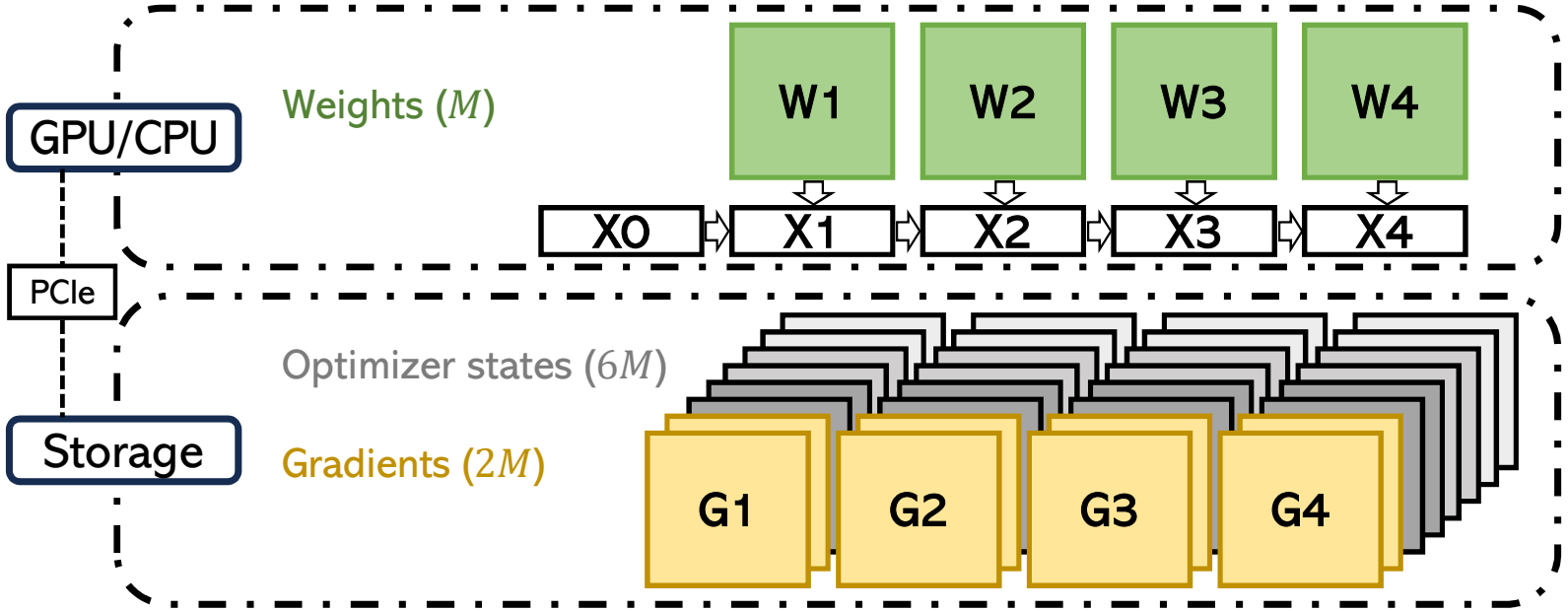
Gradients ($2M$)

Weights (M)



Storage-Offloaded Training

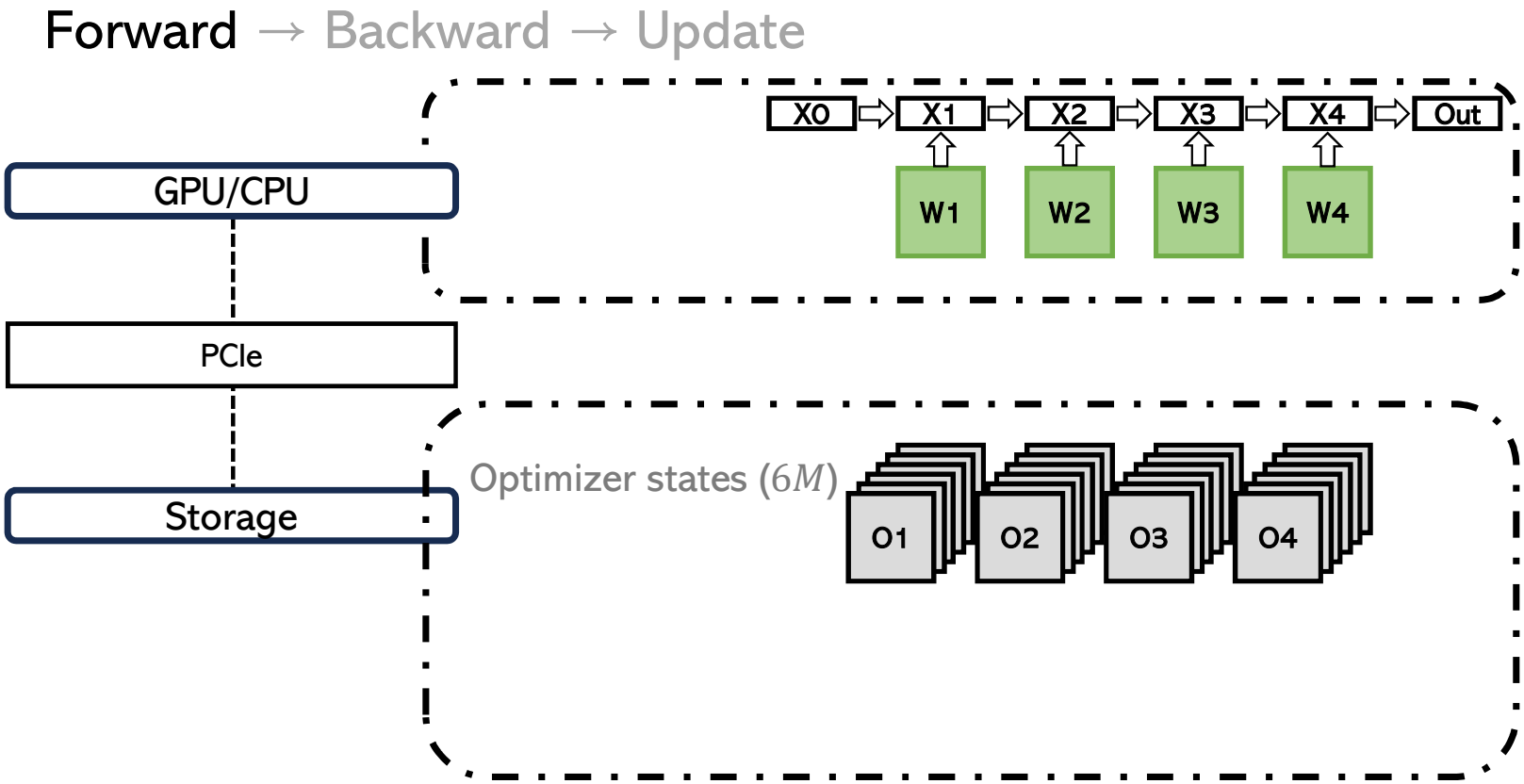
‘Storage-offloaded training’ provides flexible memory capacity for training.



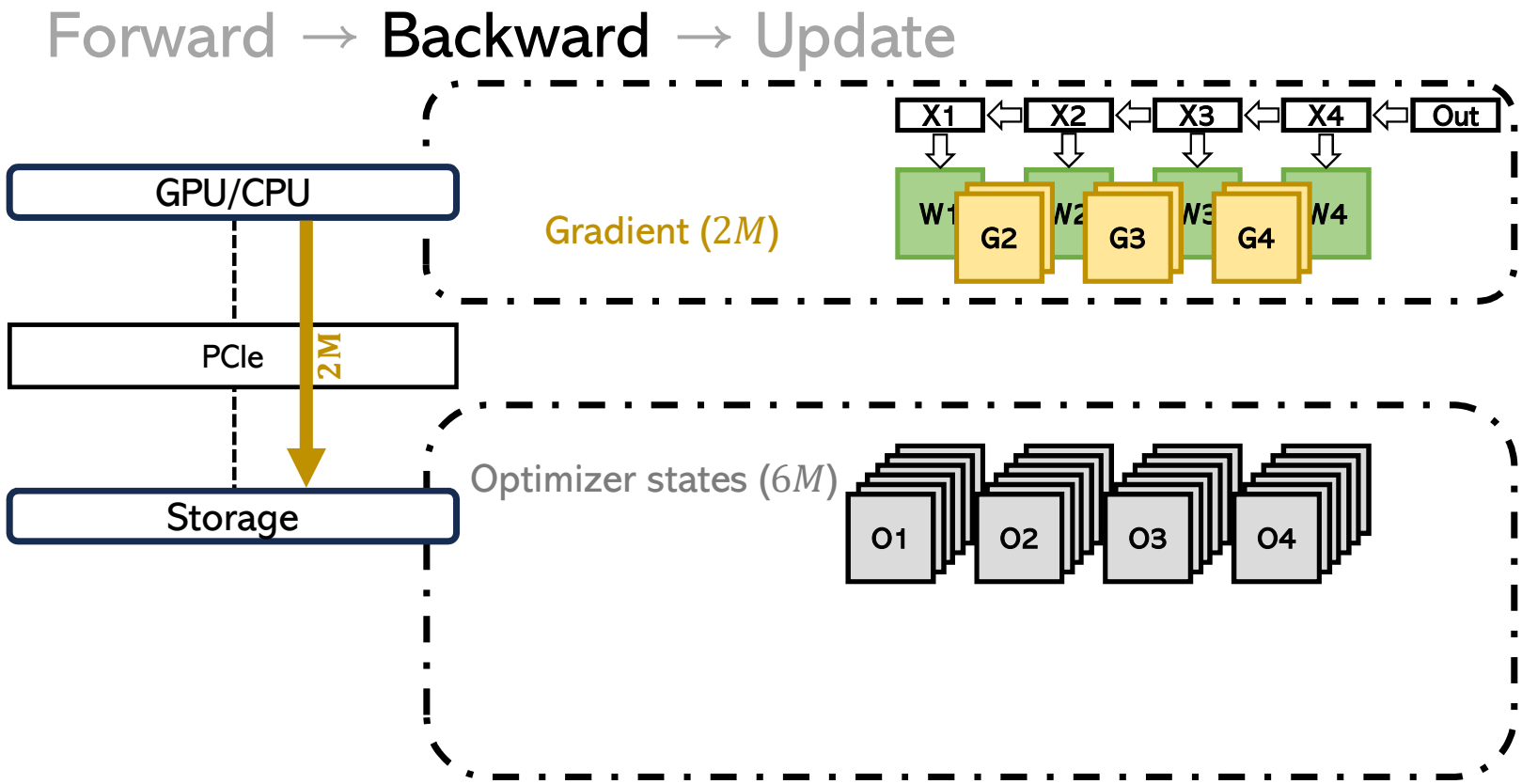
DeepSpeed

* Microsoft, “**ZeRO-Infinity**: Breaking the GPU memory wall for extreme scale deep learning”, SC’21

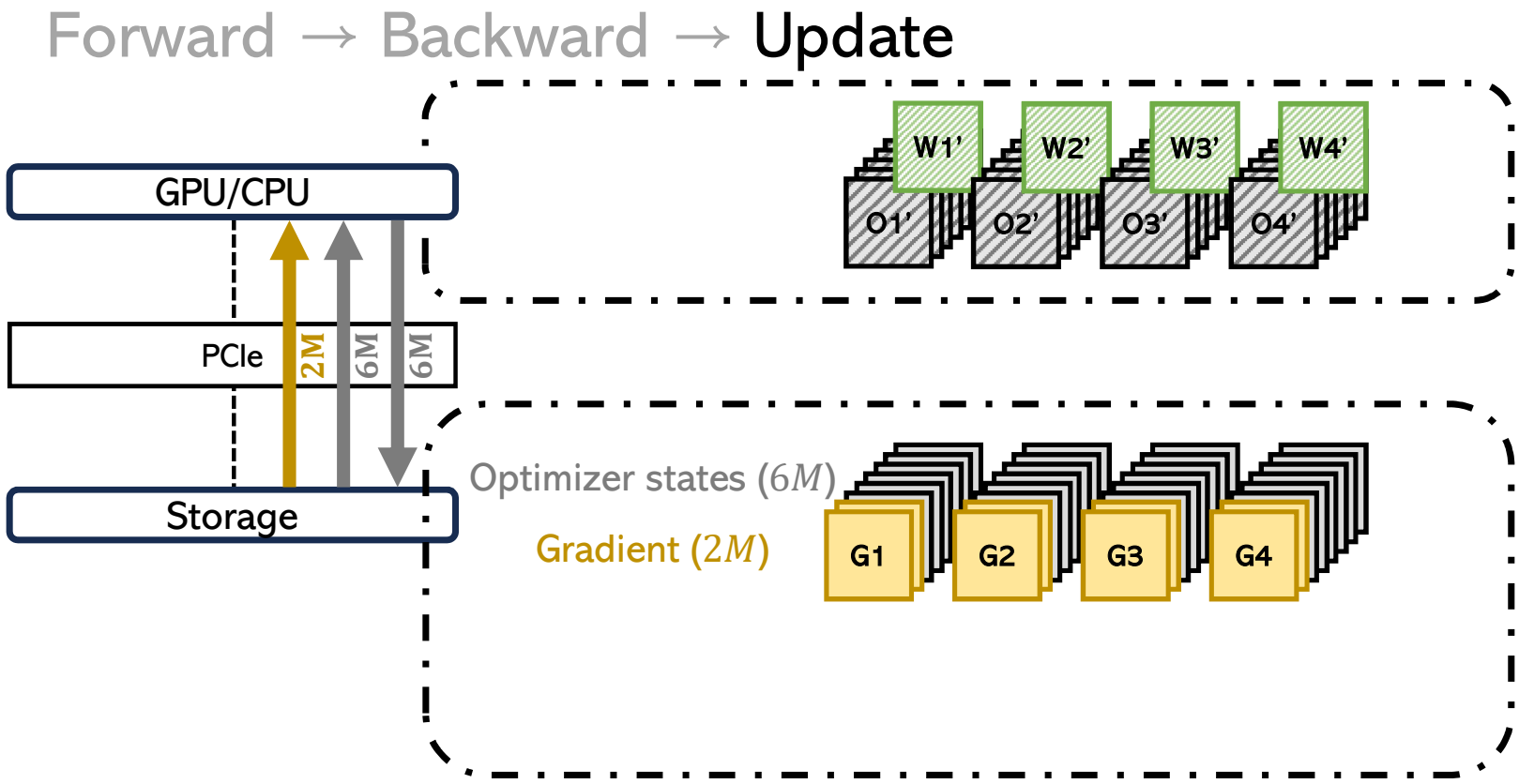
Storage-Offloaded Training (Forward)



Storage-Offloaded Training (Backward)

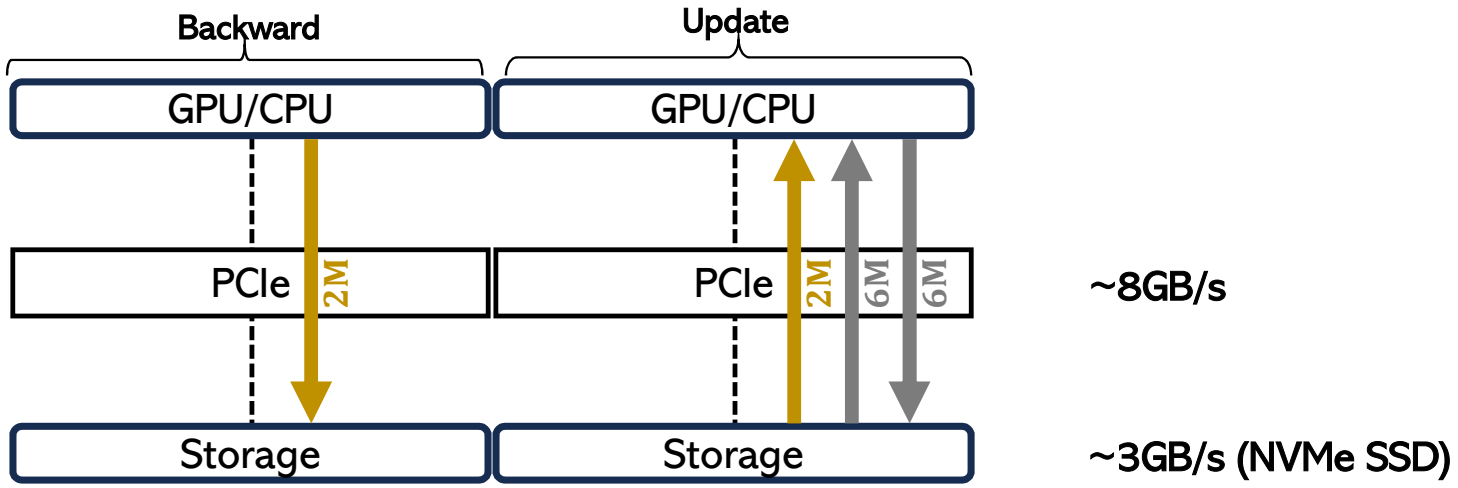


Storage-Offloaded Training (Update)



Data Transfer Overhead

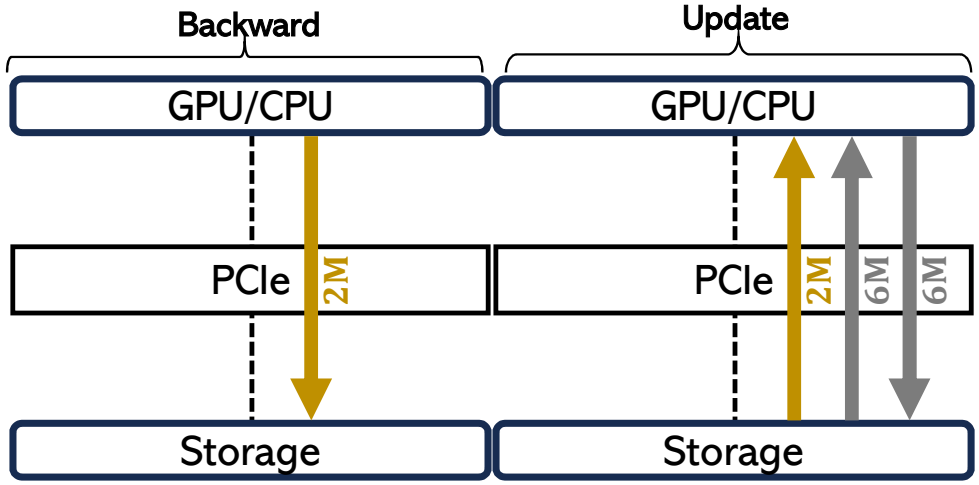
Storage traffic per single iteration

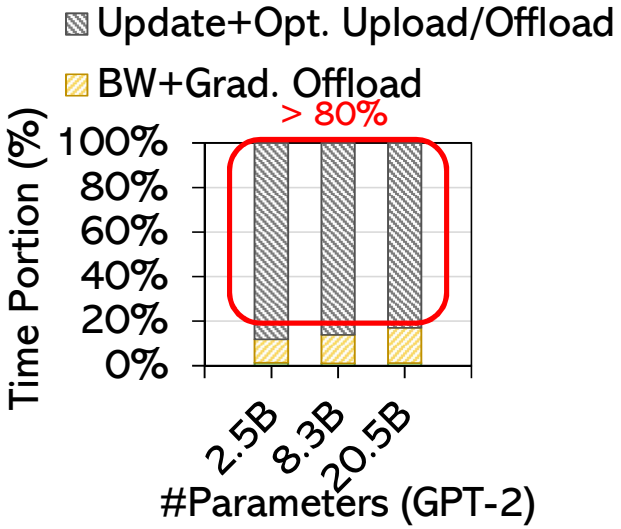


Data Transfer Overhead

Motivational experiment 1: Breakdown of storage-offloaded training time

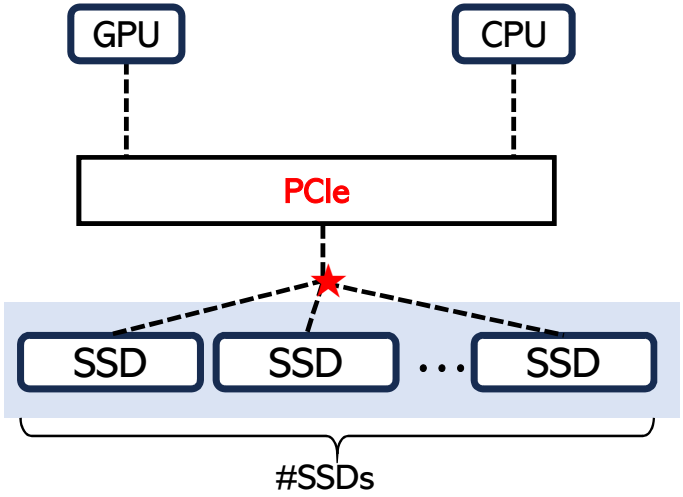
Almost no computation, mostly data transfer overhead





System Interconnect Bottleneck

More NVMe SSDs using RAID?

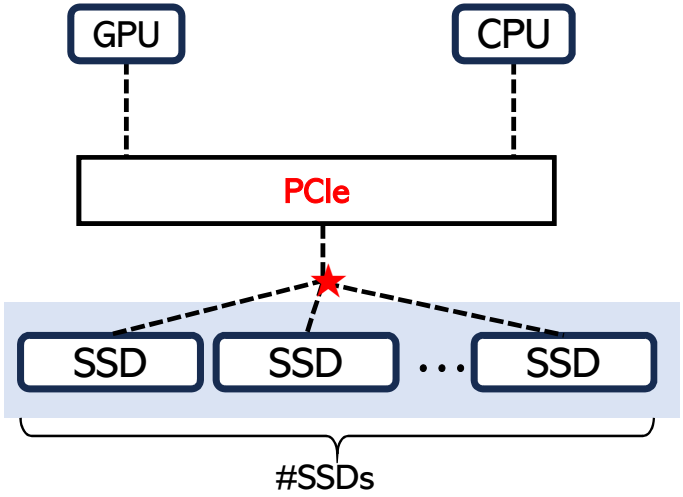


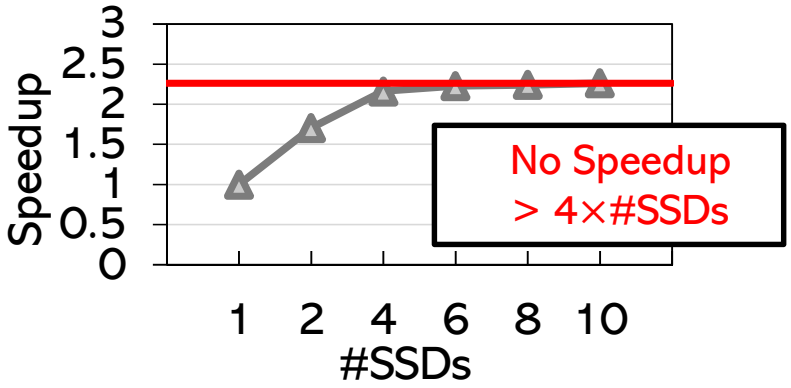
Expected aggregated bandwidth
= ~3GB/s × (#SSDs)

PCIe lanes bandwidth
= 8GB/s

System Interconnect Bottleneck

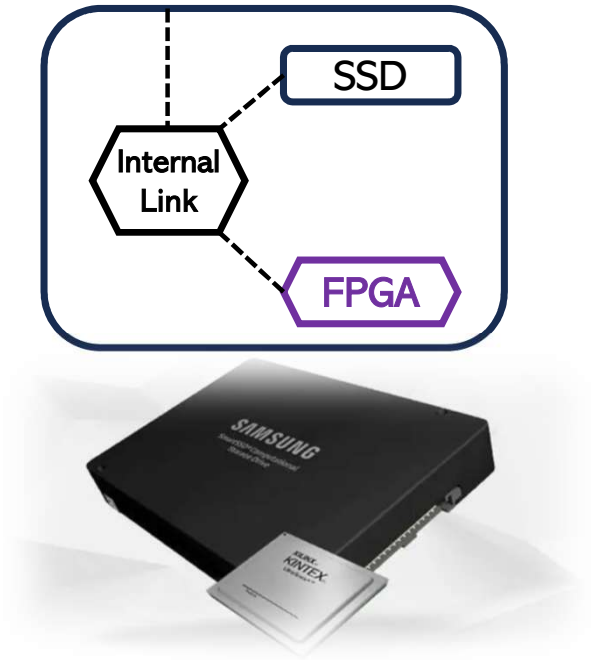
Motivational experiment 2: Increasing #SSDs using RAID0



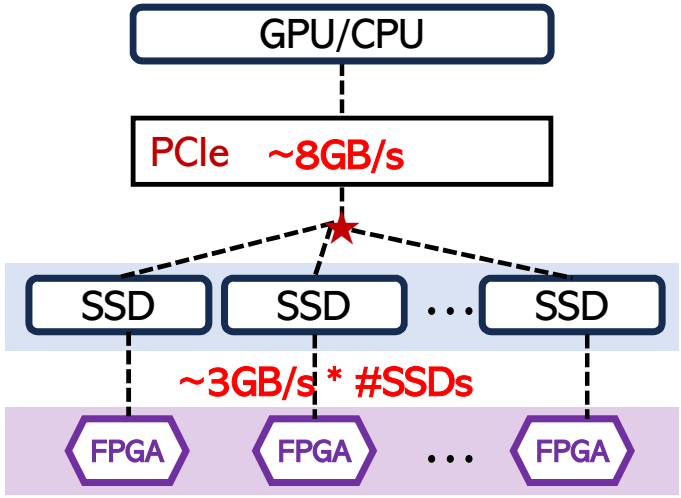


Computational Storage Devices

Computation Storage Device (CSD)
(e.g., Samsung SmartSSD)



More 'internal' bandwidth
as more SSDs are added



Overview of Smart-Infinity

Problem 1: **Data transfer overhead** of storage-offloaded training

Problem 2: **System interconnect bottleneck** limits scaling #SSDs

- **Smart-Infinity**

→ Minimizing the traffic through **shared system interconnect**

→ **Scaling #storage devices** can provide speedup



Smart-Infinity

(1) Smart-Update: Avoiding **system interconnect (PCIe) bottleneck** using **CSDs**

~~(2) Internal Data Transfer Handler: Efficient optimized structure for CSD application~~

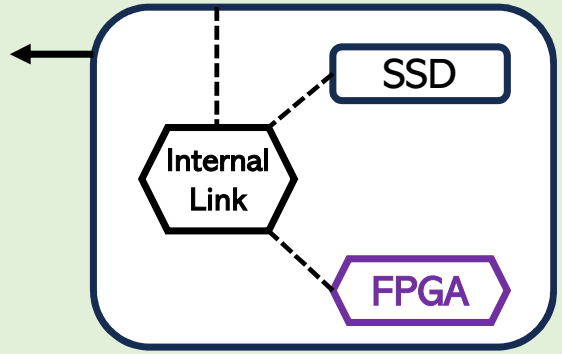
(3) Smart-Comp: **CSD**-aided compression for reducing **system interconnect traffic**



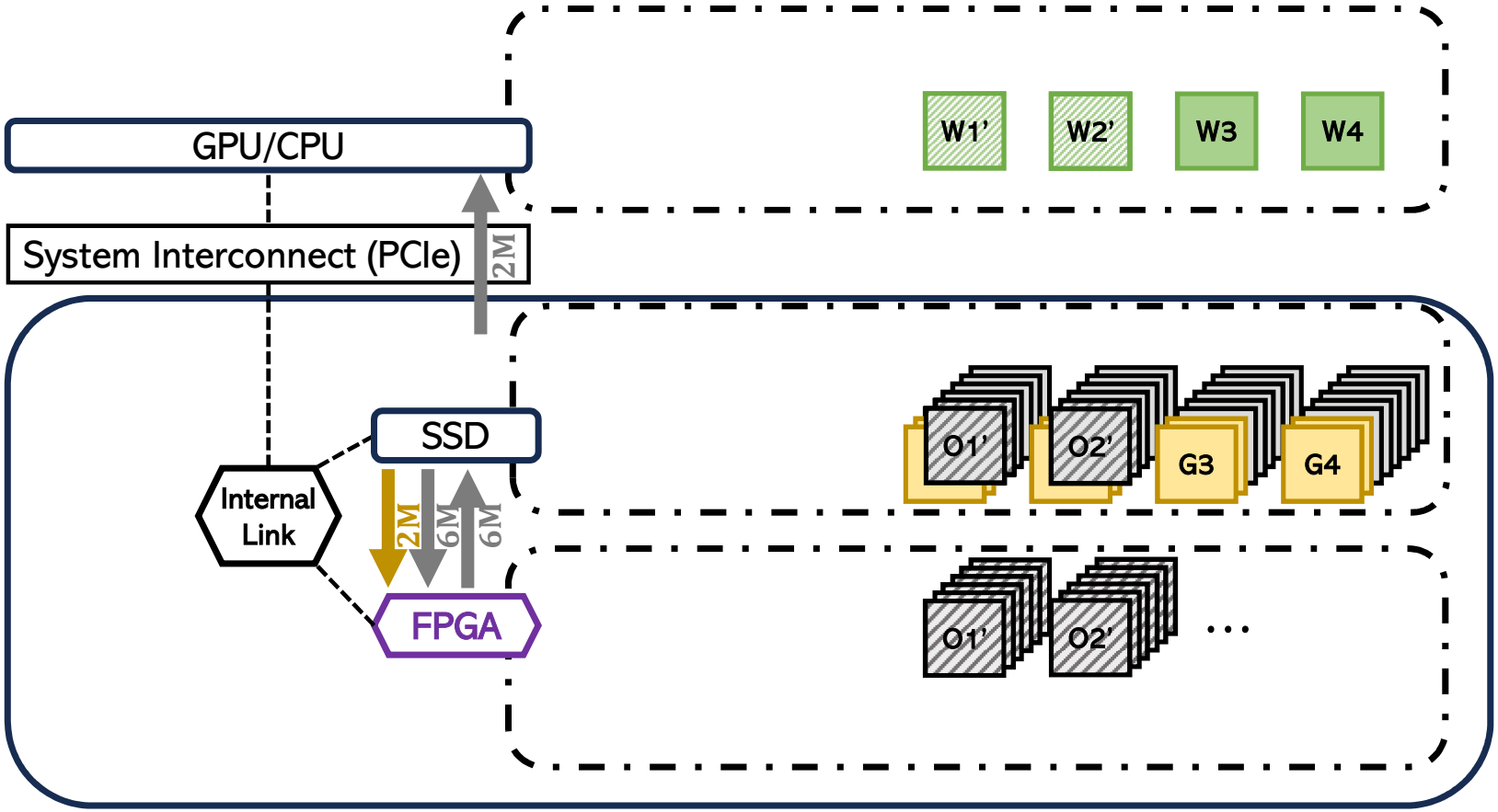
Smart-Update

(1) **Smart-Update**: Perform updates on **CSDs**

Internal bandwidth at SSD side

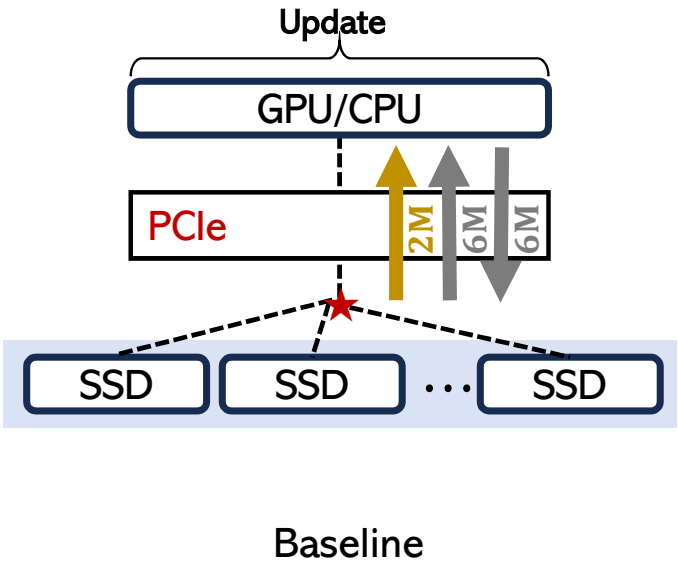


Smart-Update

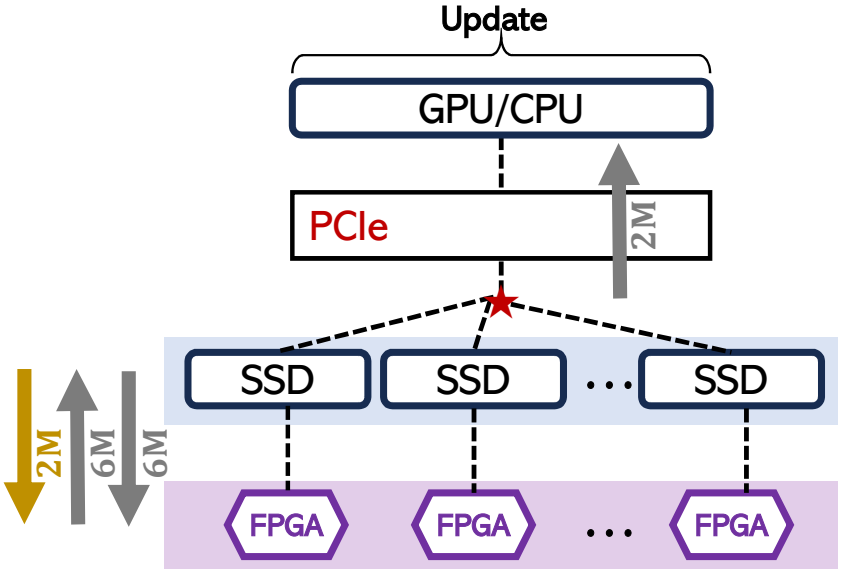


- Perform update within the FPGAs

Smart-Update



Baseline

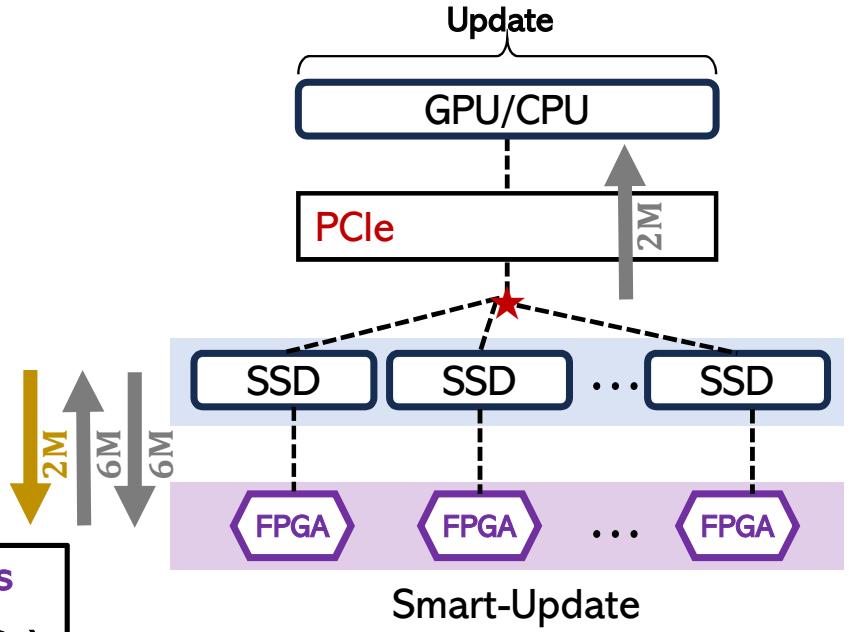


Smart-Update

Smart-Update

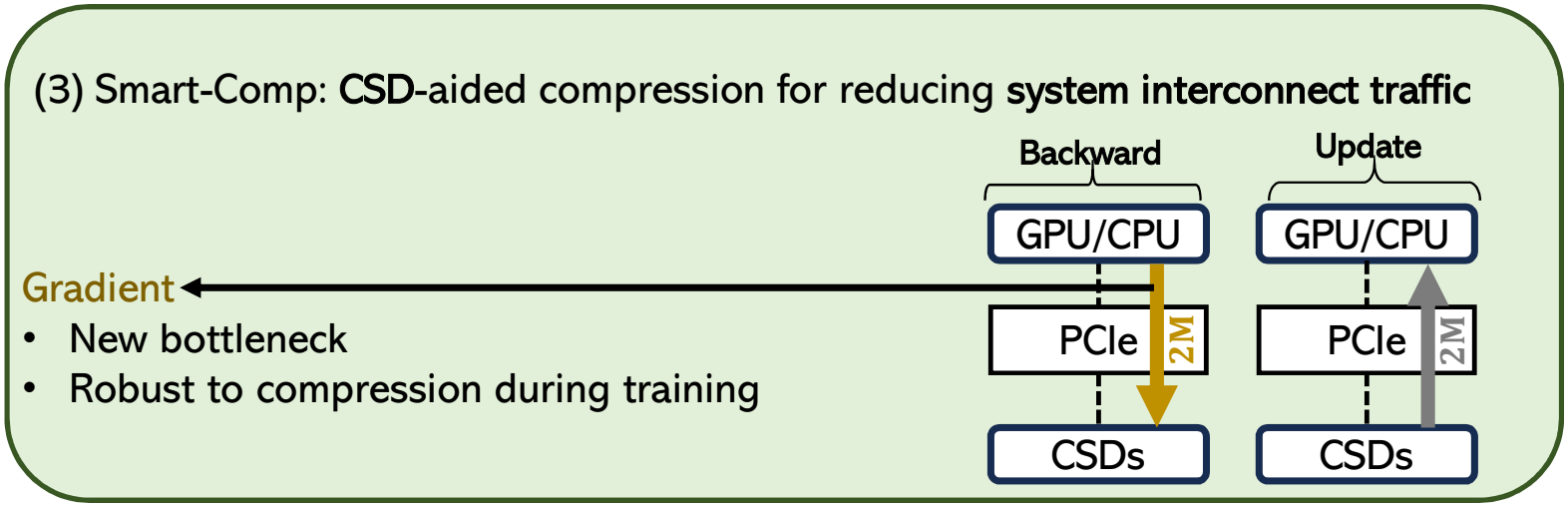
PCIe lanes bandwidth
= 8GB/s ≃ 3×(#SSDs)

Aggregated internal bandwidth of CSDs
= 3GB/s×(#CSDs) ≃ 48GB/s (w/ 16 CSDs)



Smart-Comp

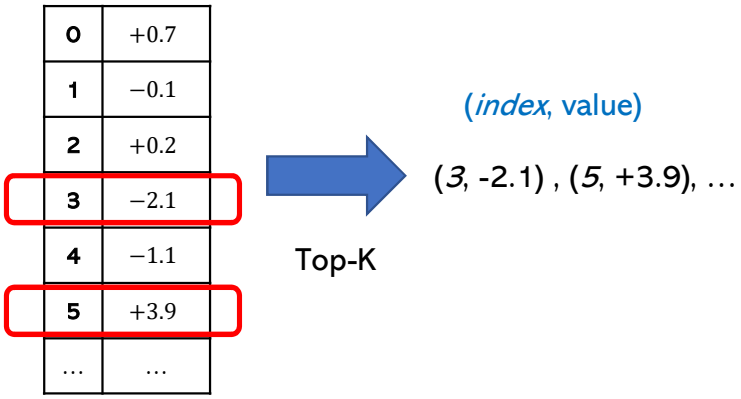
(3) Smart-Comp: **CSD**-aided compression for reducing **system interconnect traffic**

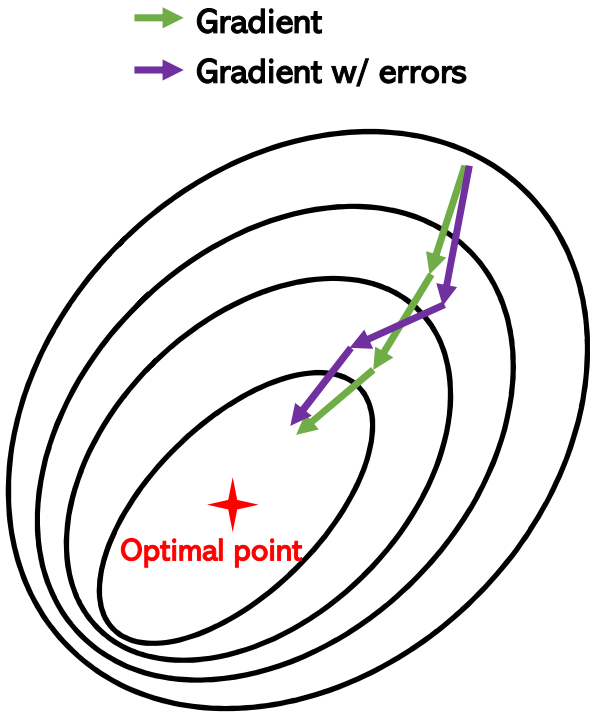


- New bottleneck
- Robust to compression during training

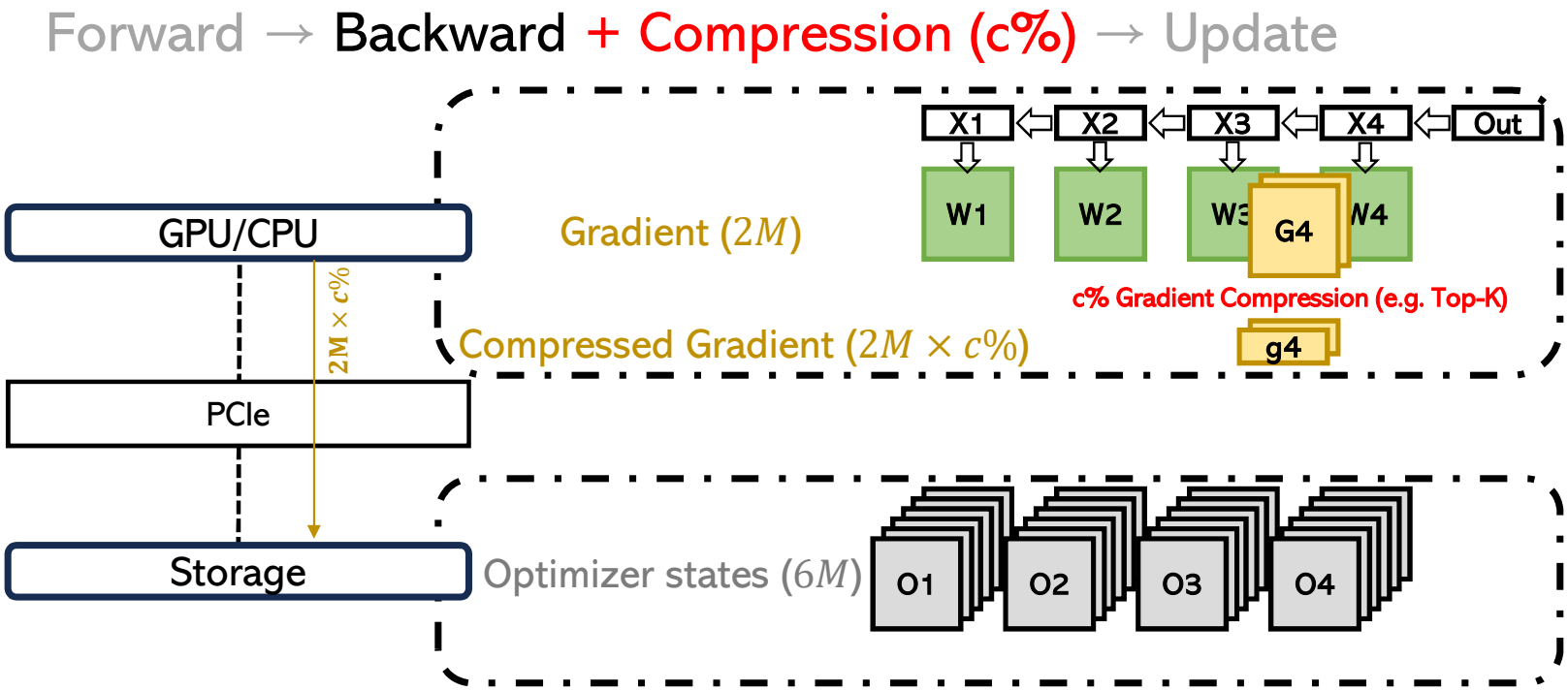
Gradient Compression Again!

- Top-K compression
- Cheaper to perform inside FPGAs
  - Compared to the low-rank ones



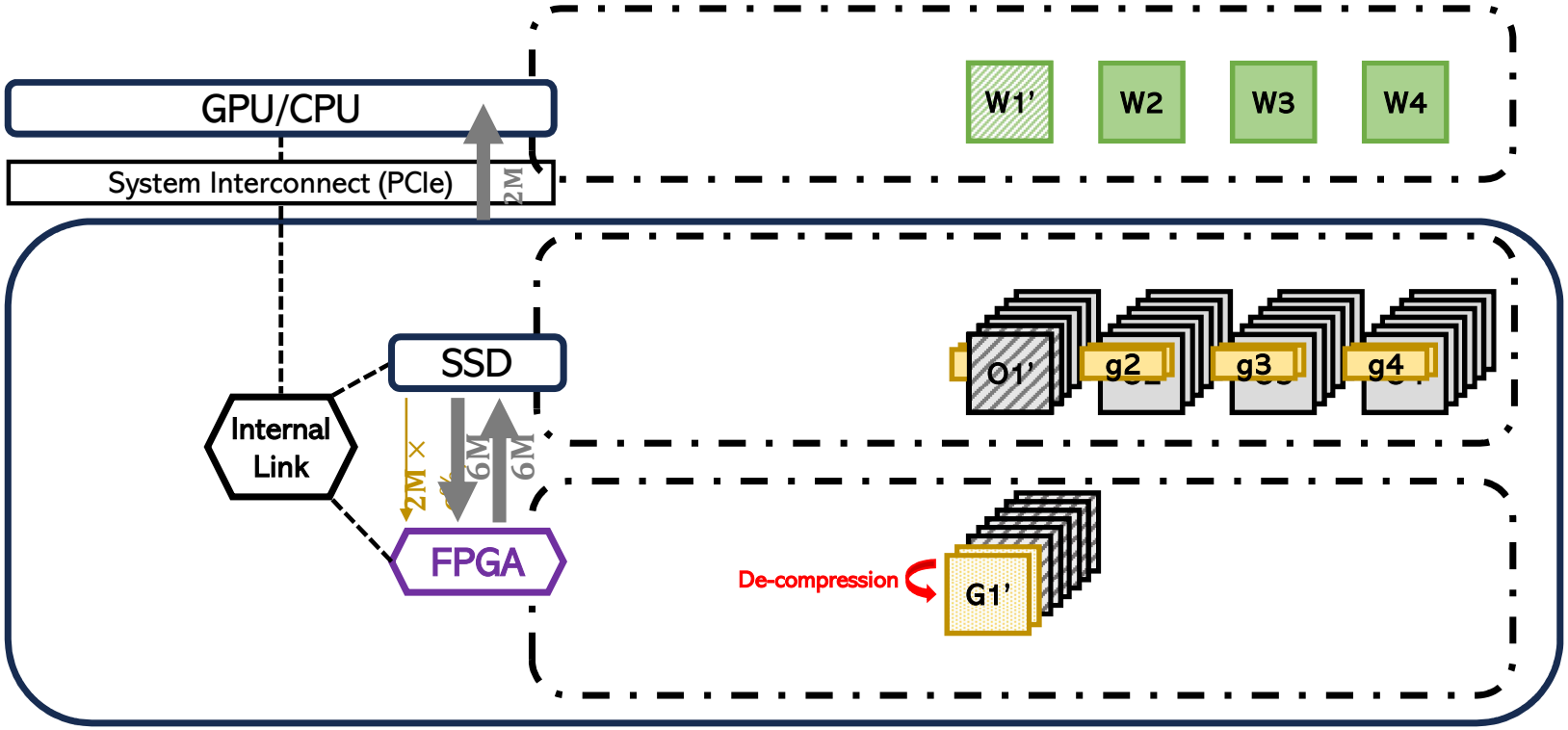


Smart-Comp (Backward)



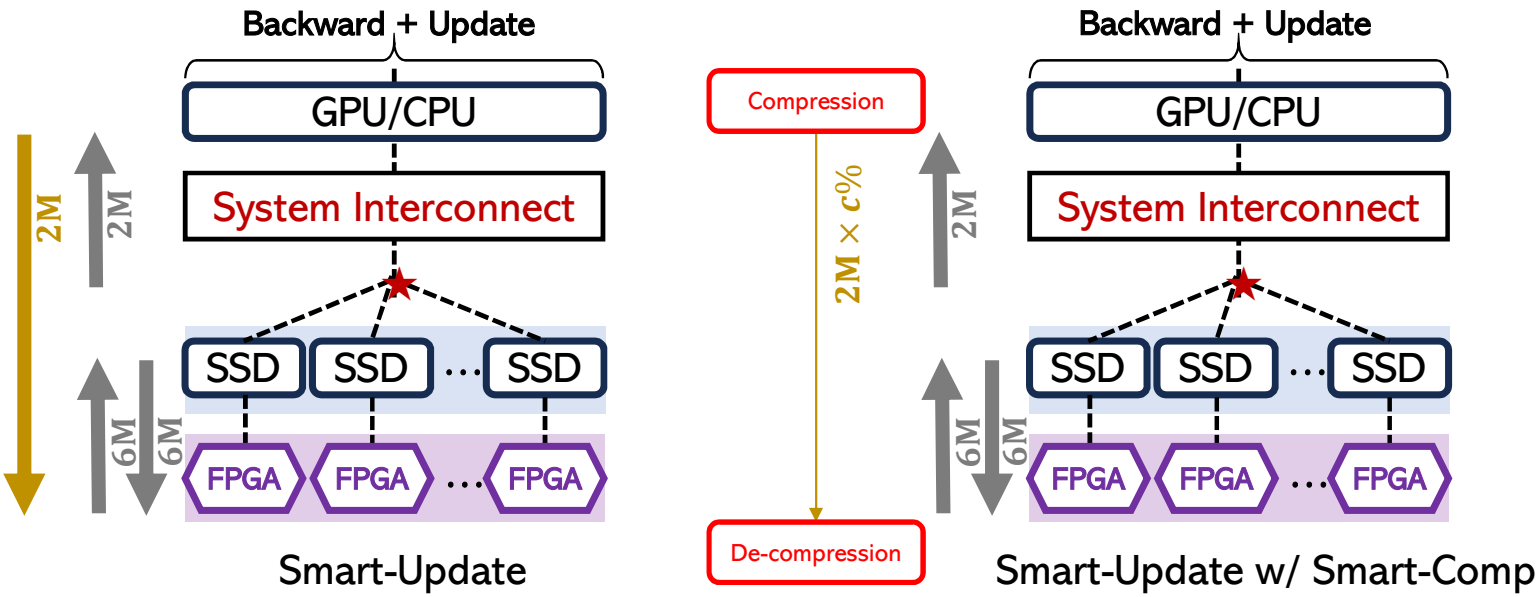
Smart-Comp (Update)

Forward → Backward → De-compression + Update

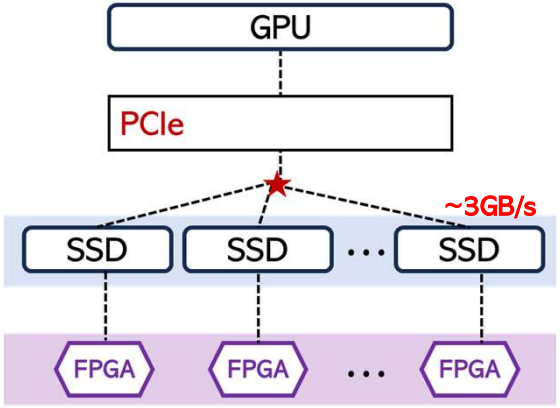


Smart-Comp

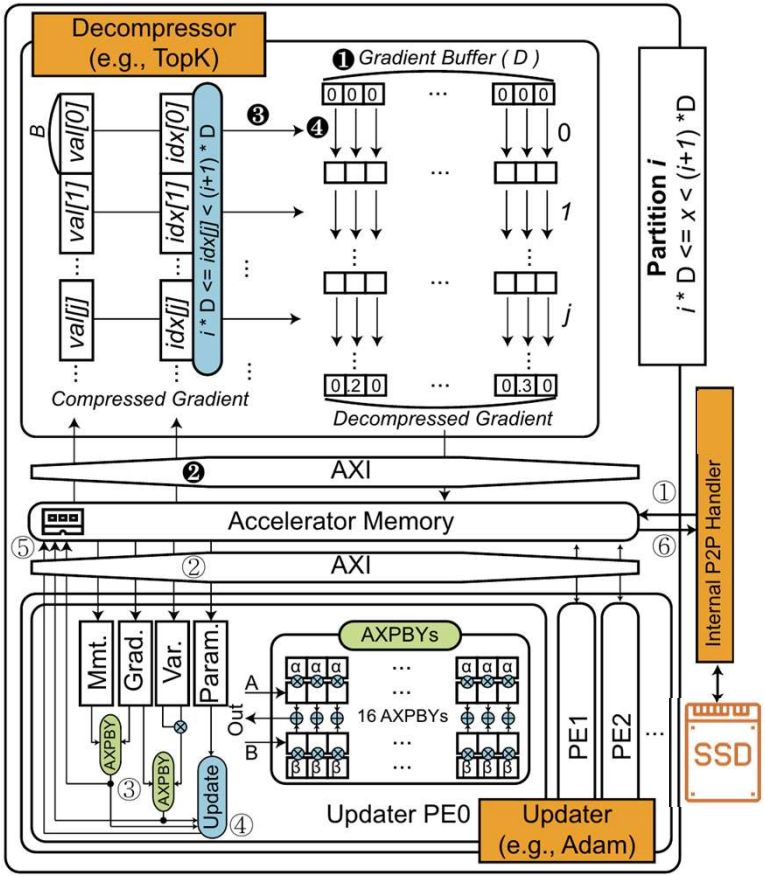
Further reducing traffic through system interconnect



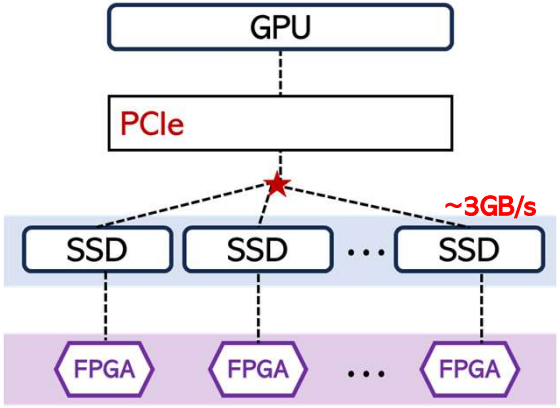
Hardware Architecture

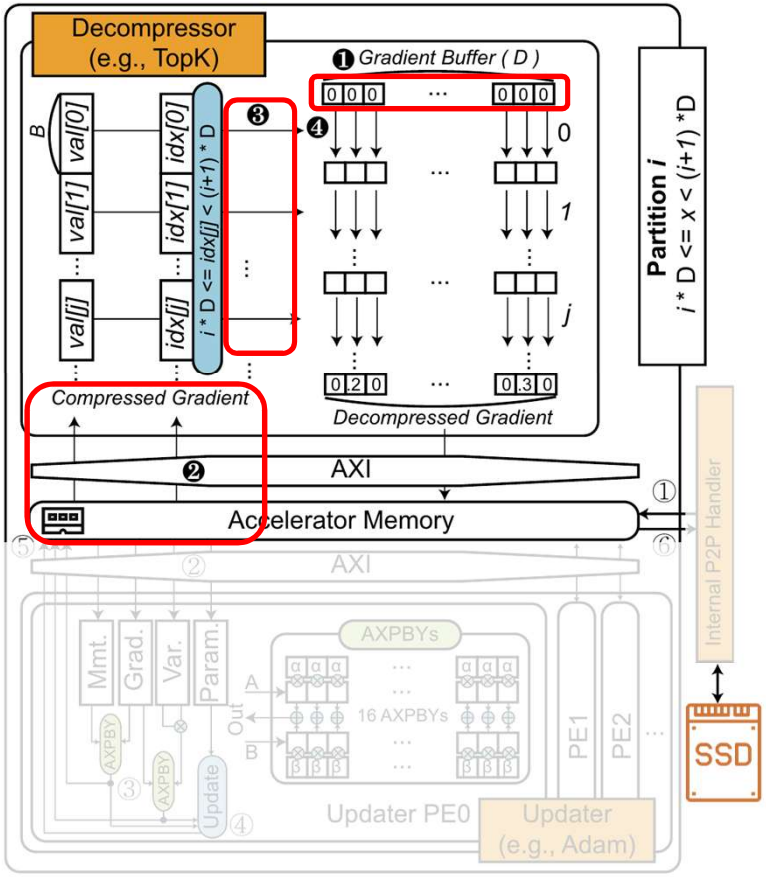


- FPGA modules designed with HLS (High-Level Synthesis)
  - Just needs to keep up with the 3GB/s SSD bandwidth per FPGA
  - Low arithmetic intensity
- Updater (Adam optimizer)
  - Weighted sum between optimizer states
  - Parallel ax*by units
- Decompressor
  - Restores Top-K gradients into a dense format



Hardware Architecture





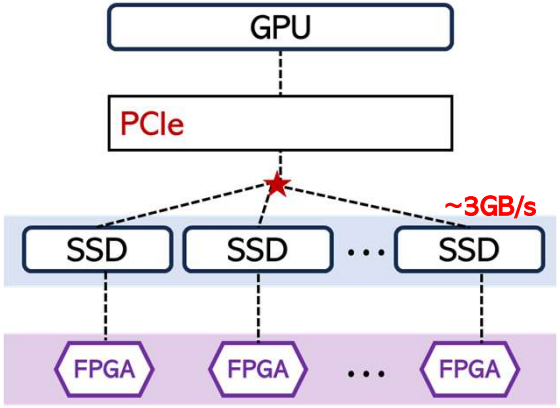
- Decompressor
  - Restores Top-K gradients into a dense format

① Initialize a dense array of '0's

② Read compressed gradients

③ Write on the array

Hardware Architecture



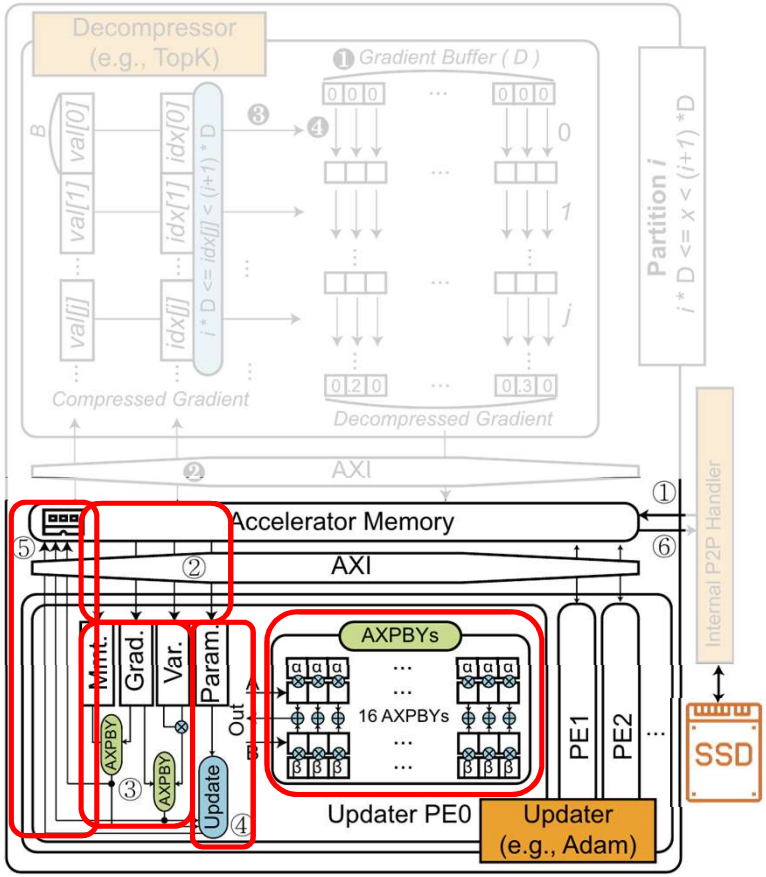
- Updater
  - Applies optimizer algorithm

①② Read data from the memory

③ Perform weighted sum

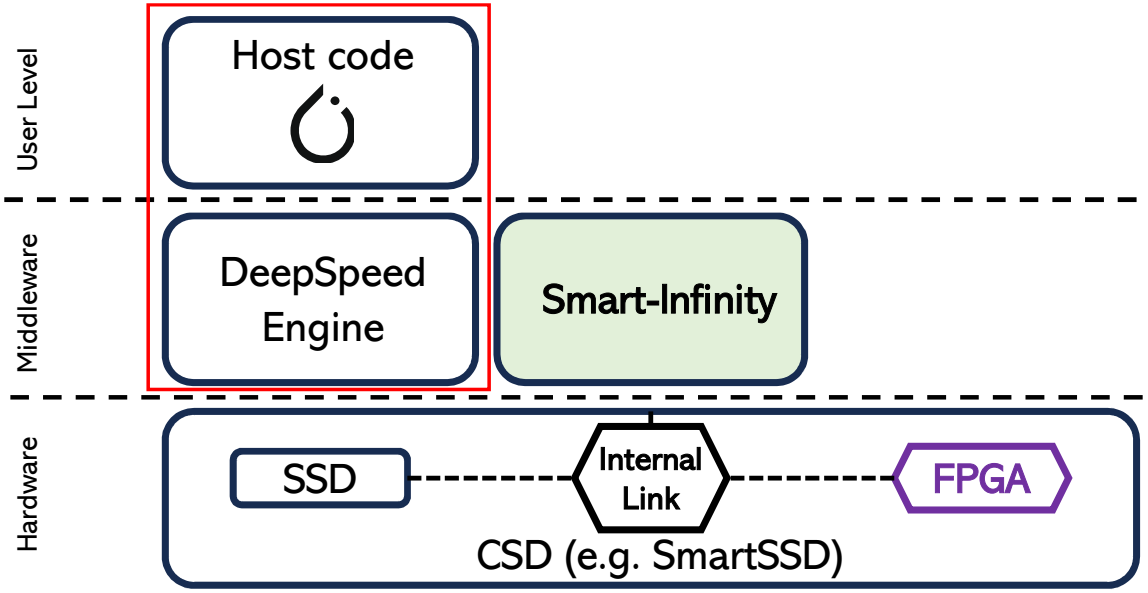
④ Update weights

⑤ Write-back

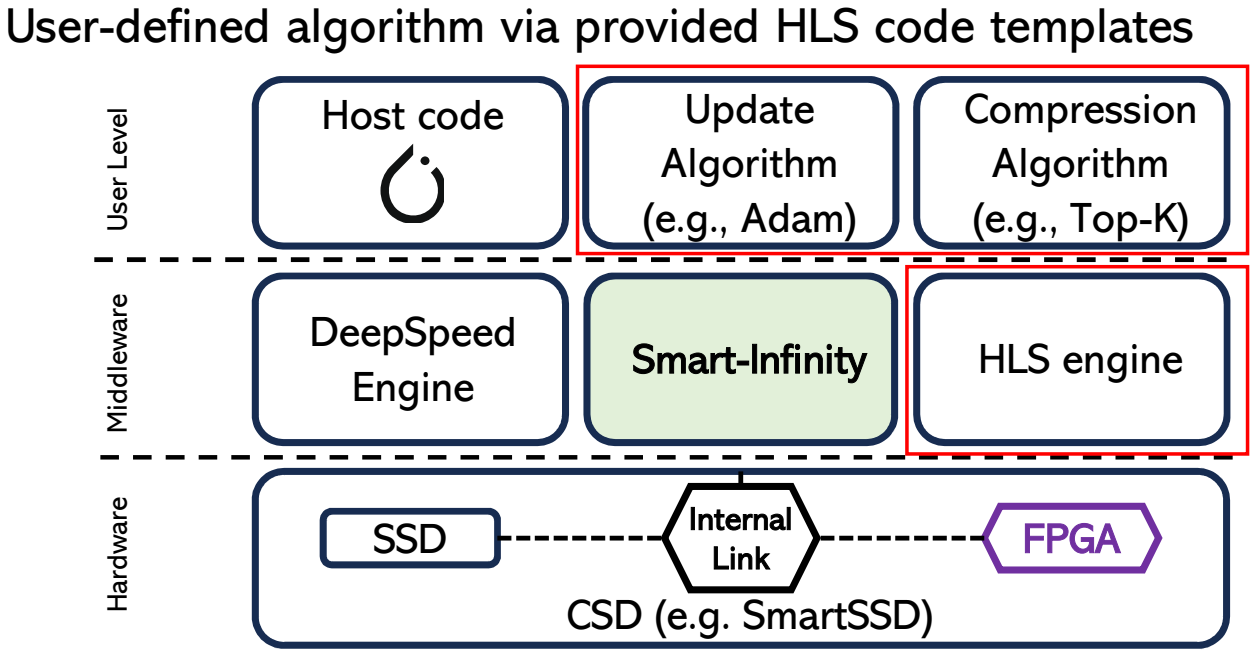


Smart-Infinity

Ready-to-use framework for storage-offloaded training



Smart-Infinity



Experimental Environment

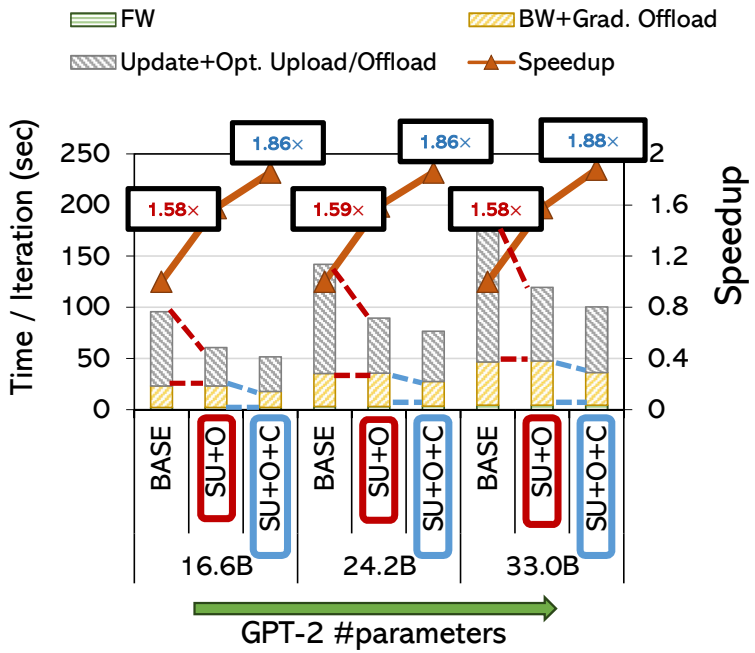
- Hardware
  - Single RTX A5000 GPU
  - SAMSUNG SmartSSD×16

- Software
  - Baseline: DeepSpeed ZeRO-Infinity
  - Magnitude-based Top-K, 2% (default)

HW	GPU	NVIDIA A5000, A100 (40GB), A4000
	CPU	Xeon(R) Gold 6342, 2×48C 96T
	Memory	32×32GB DDR4-3200
	SSD	SAMSUNG SmartSSD, 4TB
	PCIe Expansion	H3 Falcon 4109
SW	OS	Ubuntu 20.04 LTS
	Python / PyTorch	3.9 / 1.12.1
	CUDA / OpenCL	11.6.2 / 2.2
	Vitis / XRT	2023.1 / 2.21.427
	Model	GPT-2, BERT, BLOOM, ViT
	Deepspeed	0.9.3



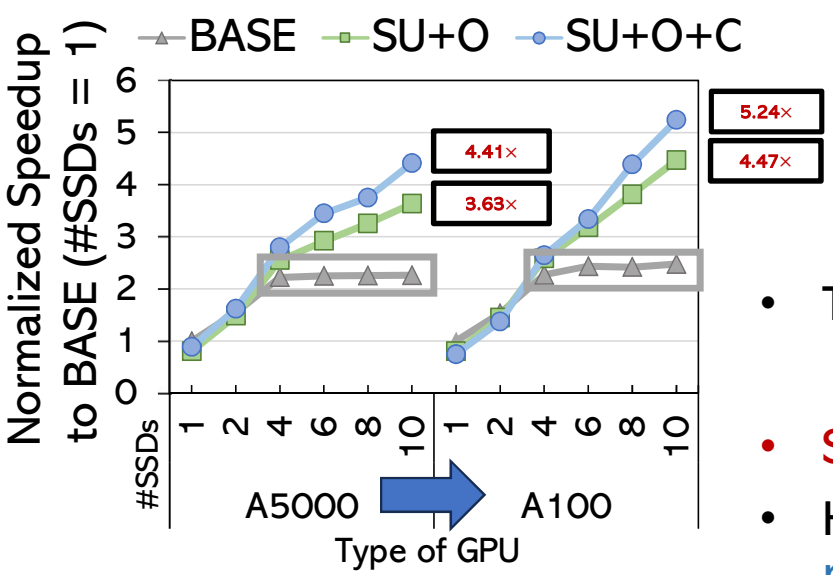
Main Results & Model Size Scalability



Notations	
BASE	Baseline with RAID0 (#SSDs=16)
SU+O	Smart-Update w/ Optimizations
SU+O+C	SU+O+Smart-Comp (2%)

- From reduced optimizer states
- From further reduced gradient
- Significant 1.5~1.9x speedup
- Scalable on larger models (⟶)

#SSDs Scalability & GPU Type Sensitivity

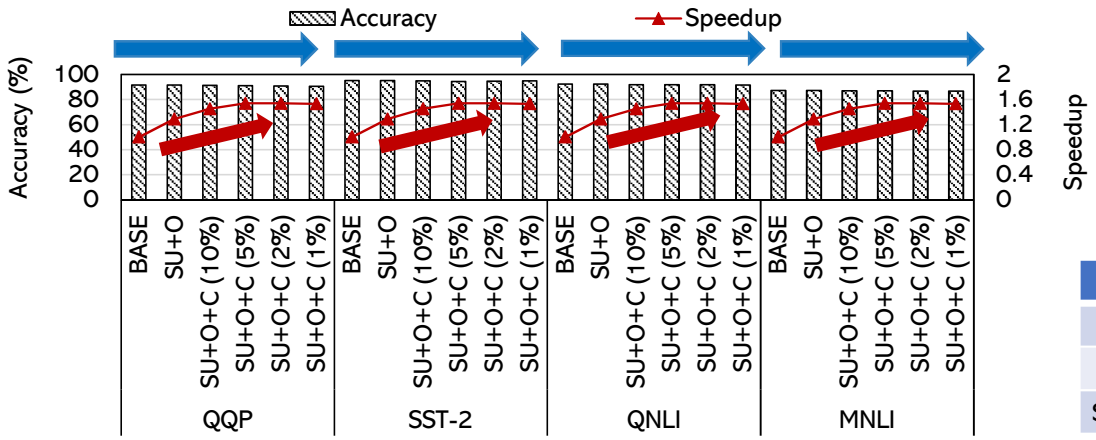


Notations	
BASE	Baseline with RAIDO
SU+O	Smart-Update w/ Optimizations
SU+O+C	SU+O+Smart-Comp (2%)

- The baseline has limited scalability.
- Smart-Infinity is scalable.
- Higher scalability w/ more compute power (⟹)

Compression Ratio Sensitivity

- Smart-Update (SU+O) **does not change the algorithm at all.**
- Smart-Comp (SU+O+C) shows comparable accuracies with steady speedups.



Notations	
BASE	Baseline with RAID0 (#SSDs=6)
SU+O	Smart-Update w/ Optimizations
SU+O+C	SU+O+Smart-Comp



SmartSSD – What do we do?



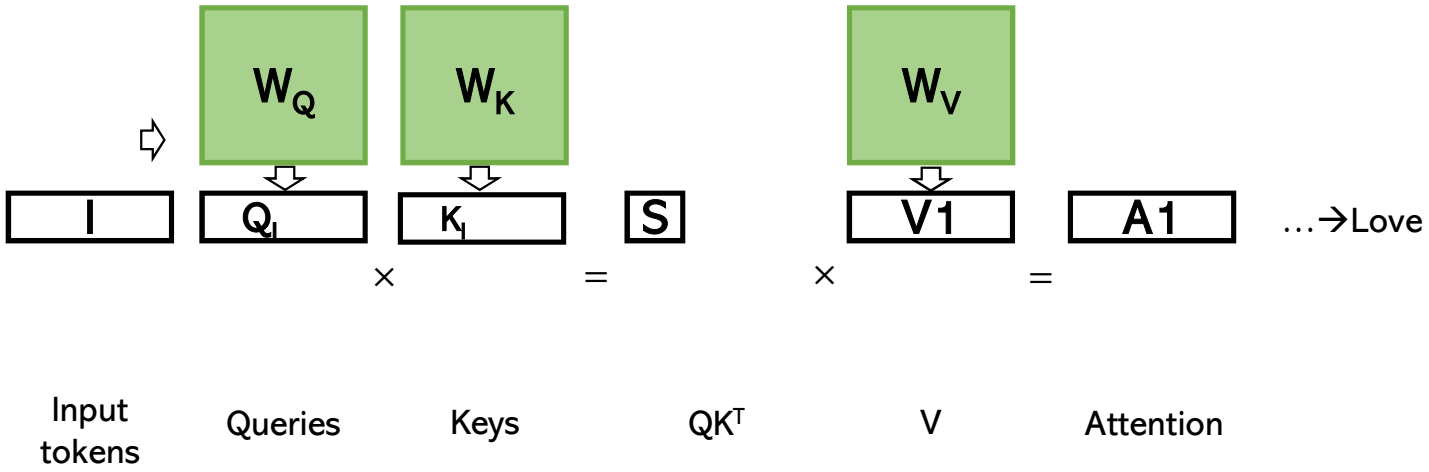
- LLM Training
  - H. Jang+, "Smart-Infinity", HPCA 2024
    - Best paper honorable mention
  - Target gradient descent
  - Put optimizer states to SSD
  - Apply gradient compression for more speedup

- LLM Inference
  - H. Jang+, "HILOS", ASPLOS 2026
  - Target multi-head attention
  - Put KV cache to SSD
  - System-level optimizations for more speedup



The LLM Inference

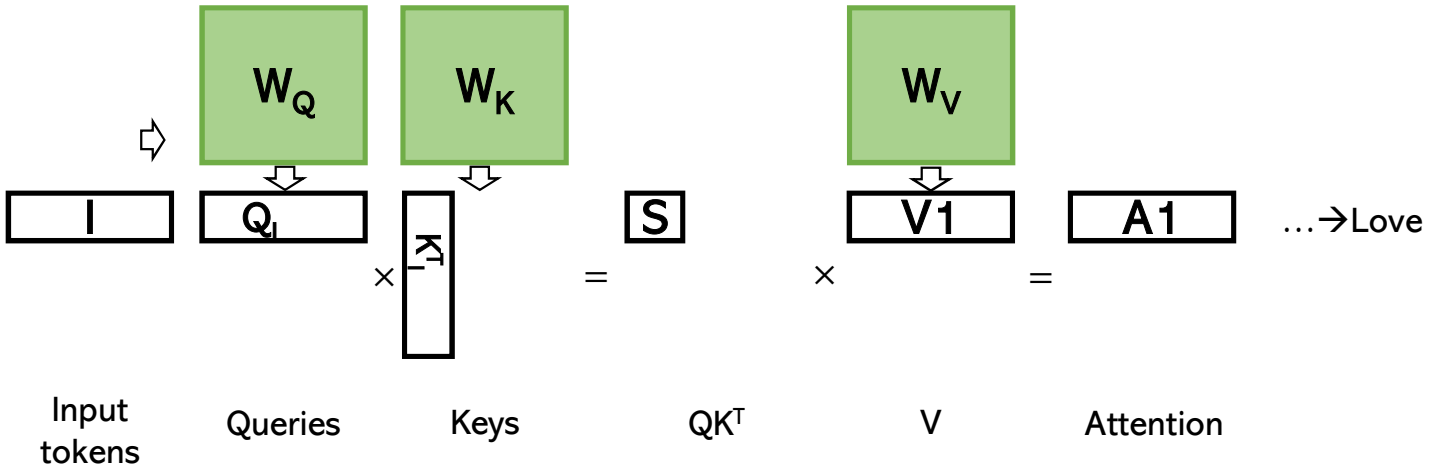
Multi-head Attention (Generating “I love PIM”)





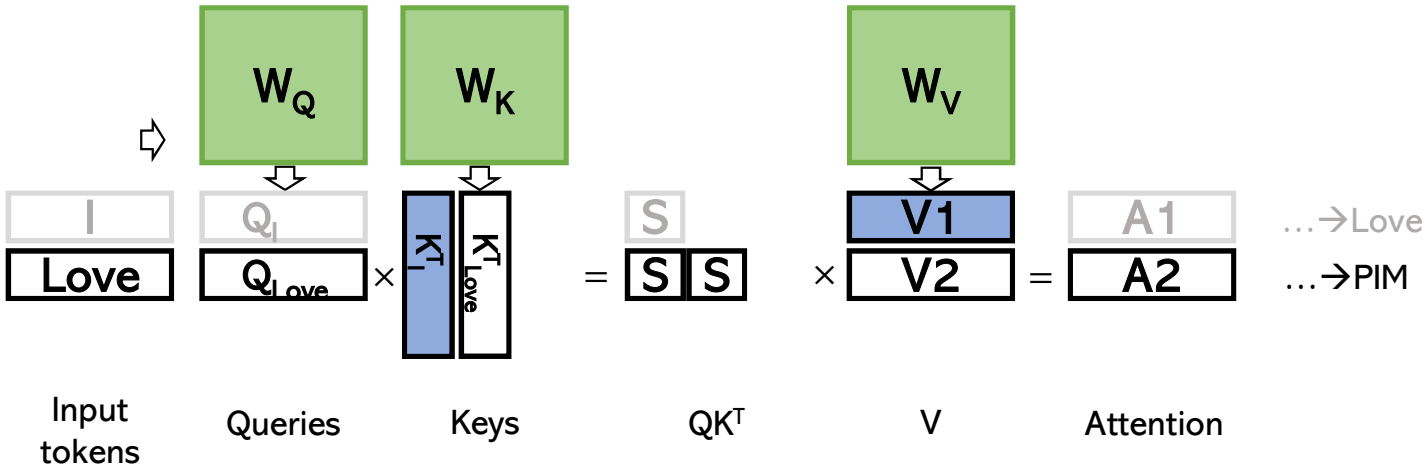
The LLM Inference

Multi-head Attention (Generating “I love PIM”)



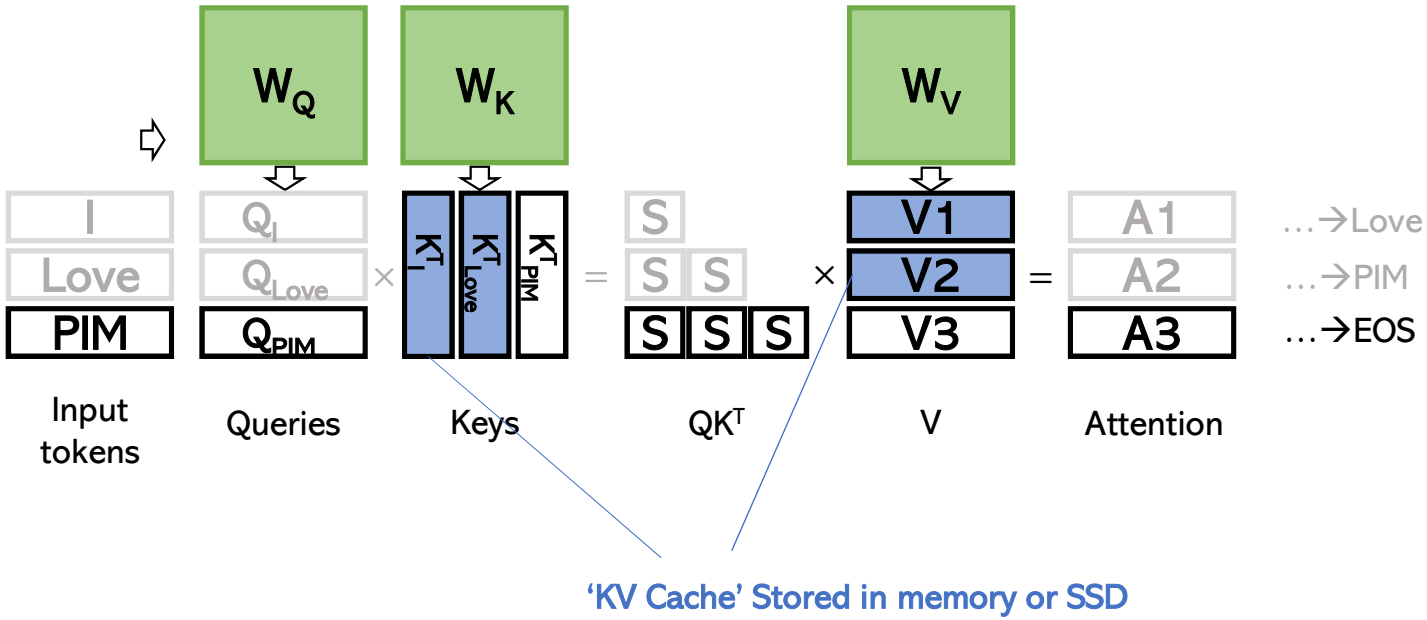
The LLM Inference

Multi-head Attention (Generating “I love PIM”)



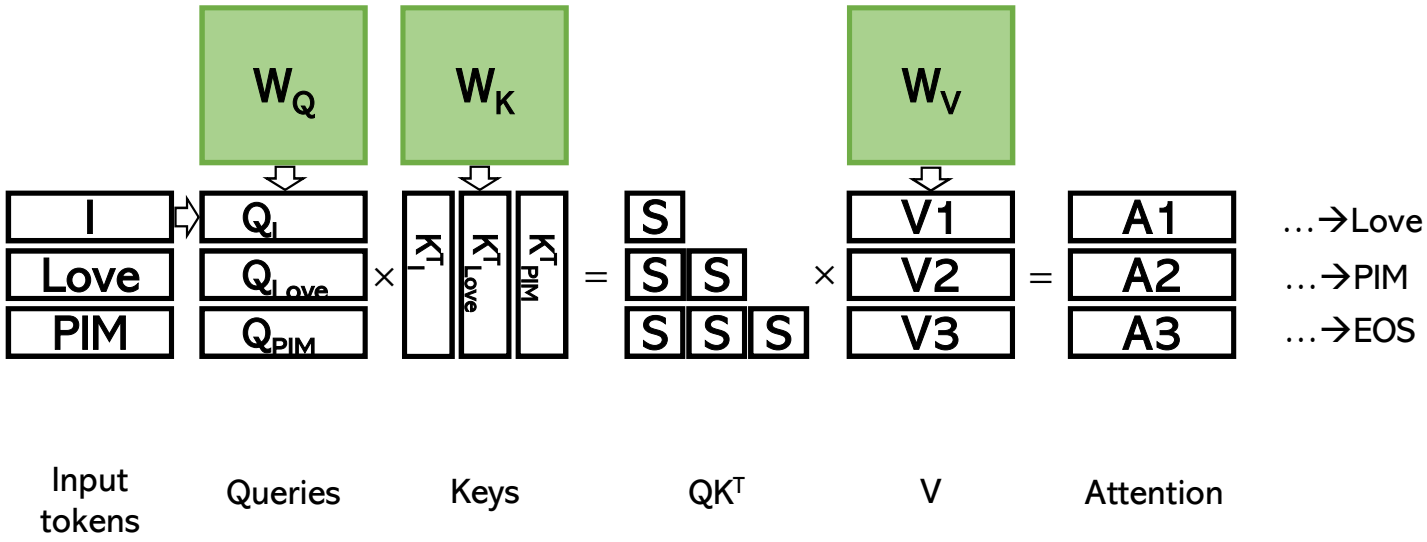
The LLM Inference

Multi-head Attention (Generating “I love PIM”)

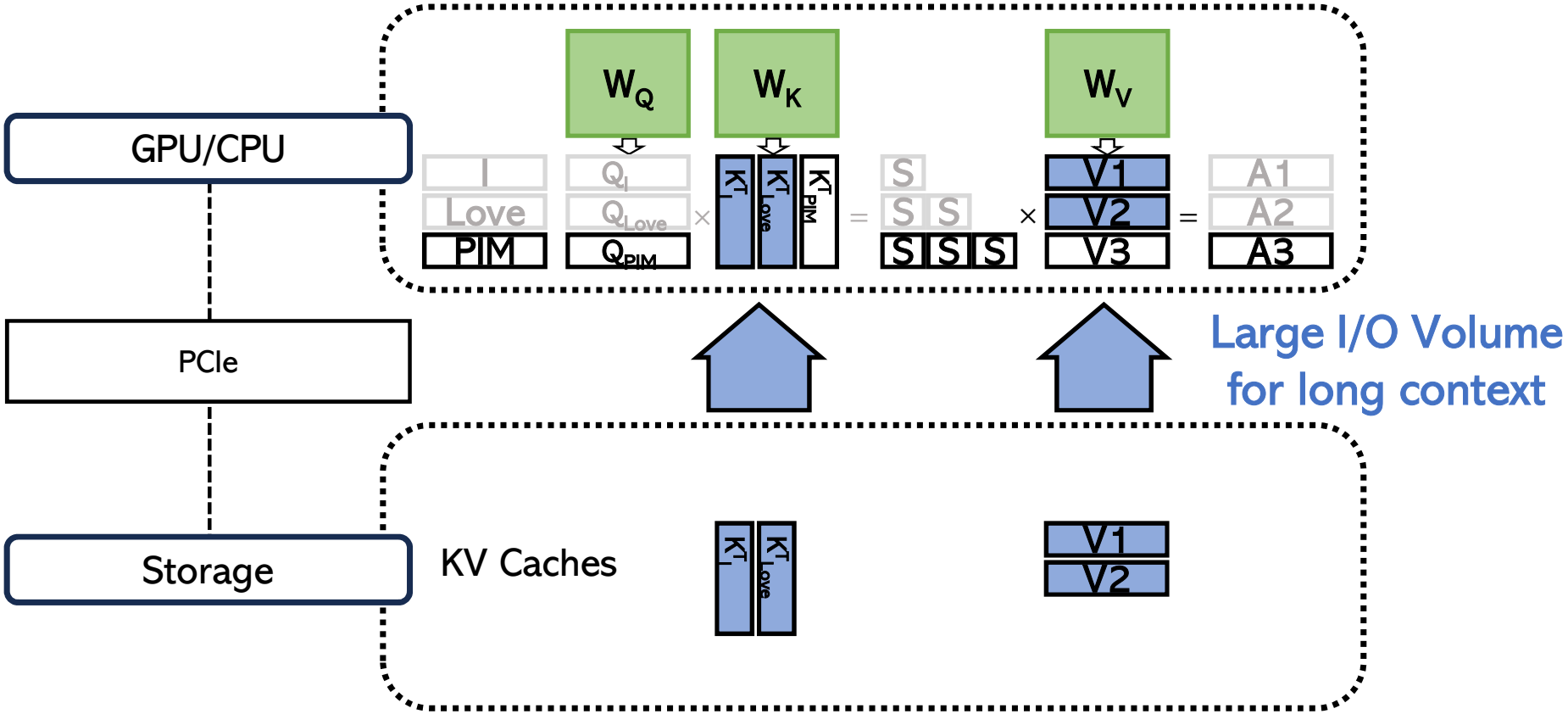


The LLM Inference

Multi-head Attention (Generating “I love PIM”)



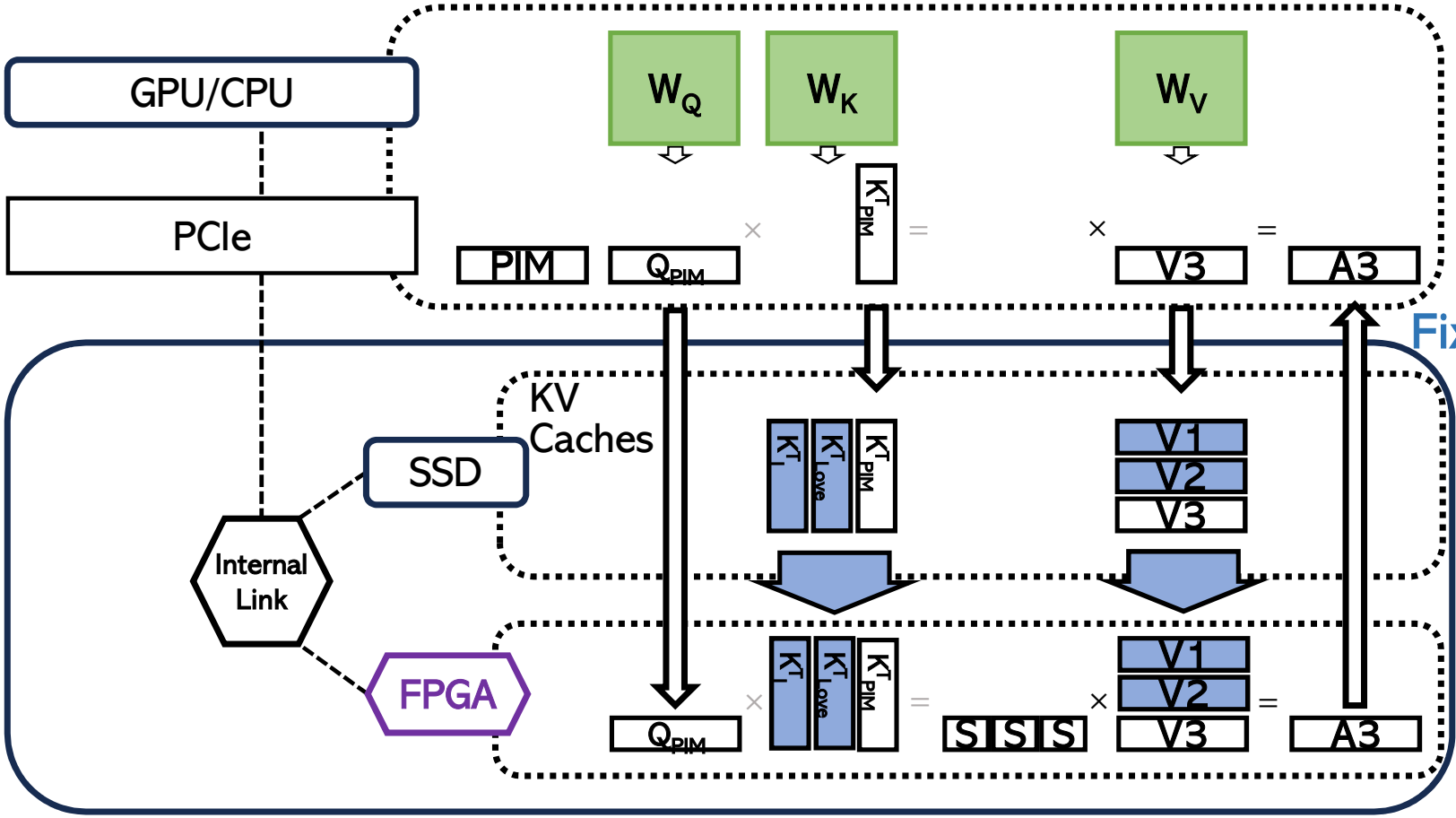
Storage-Offloaded Inference



- The sequence length can go up to a few thousand to millions
  - And is already ‘prefill’ed with long user prompt
- KV caches are stored in SSDs

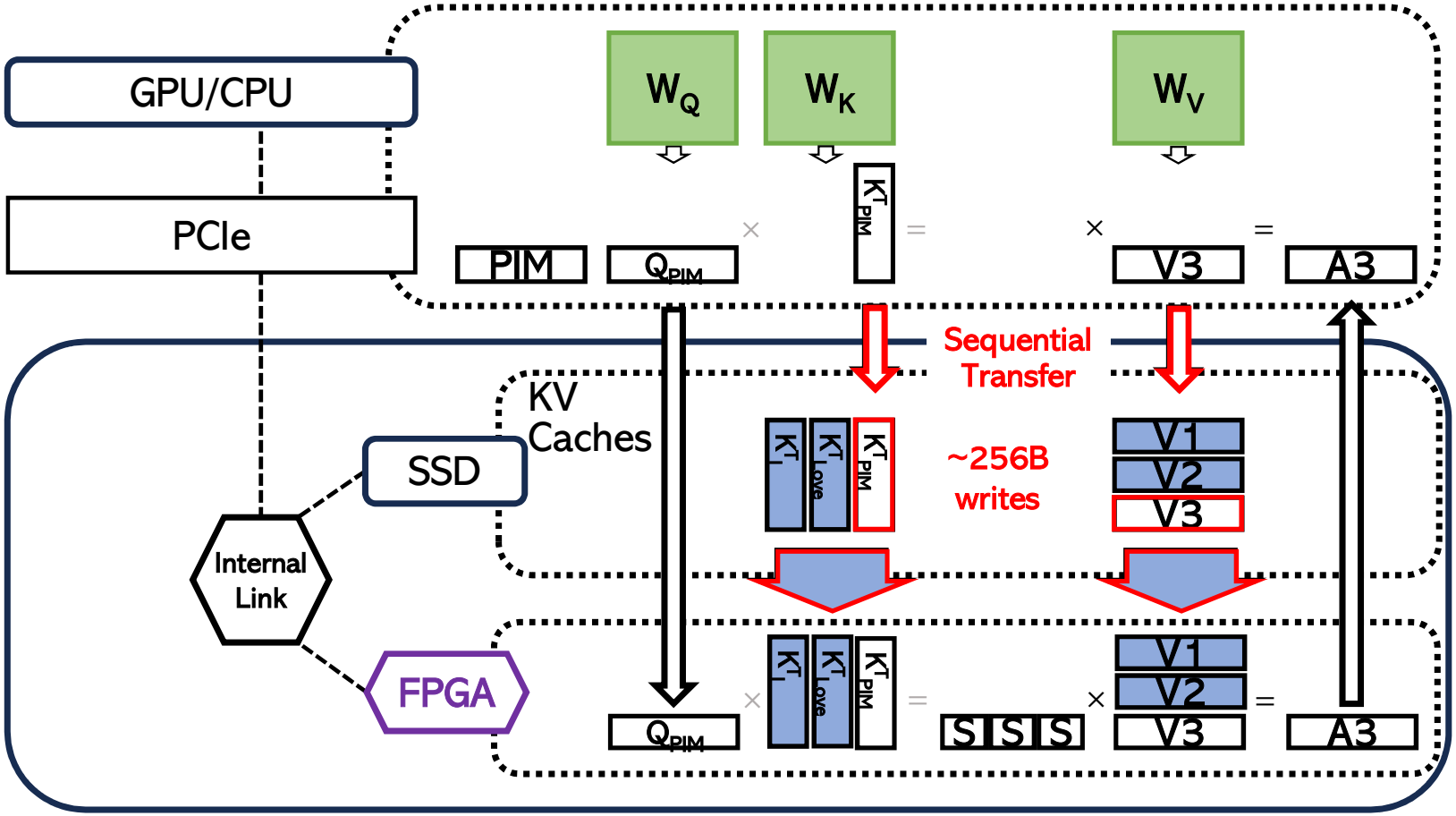


Attention in Storage (ANS)



- Small amount of computation in FPGA
- Small data transfer (does not increase with context length)

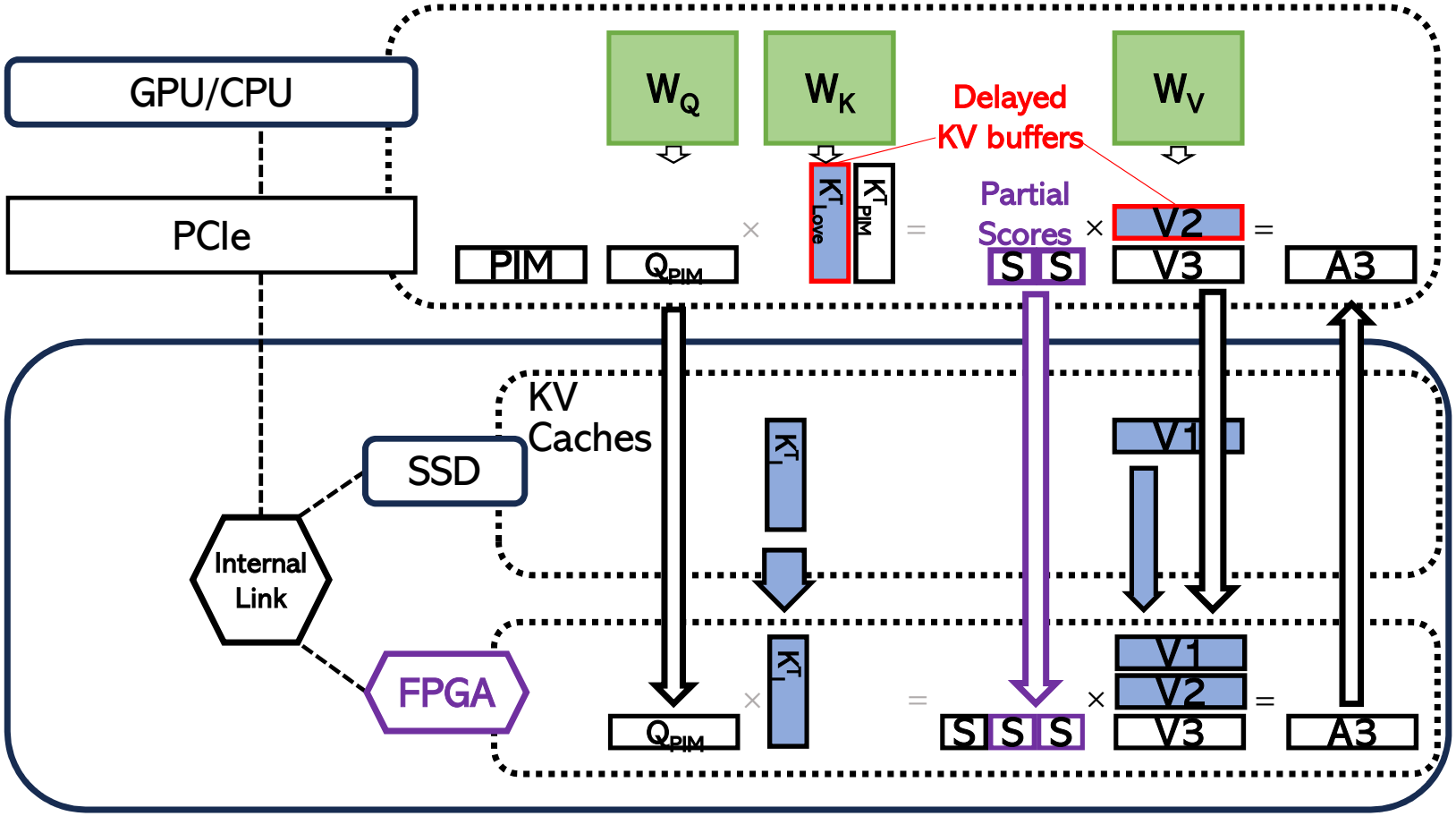
Delayed Writeback (WB)



- Sequential write causes write-induced stalls
- Small write granularity is inefficient for SSDs

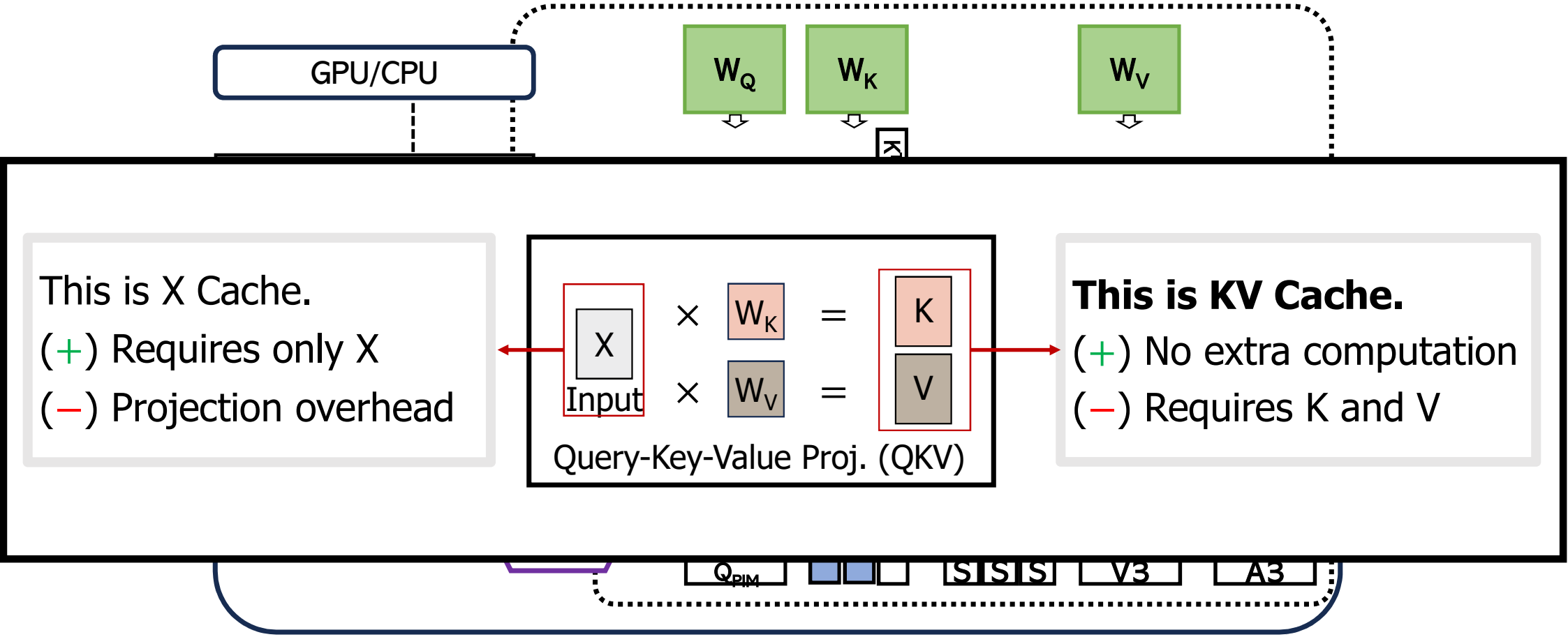


Delayed Writeback (WB)



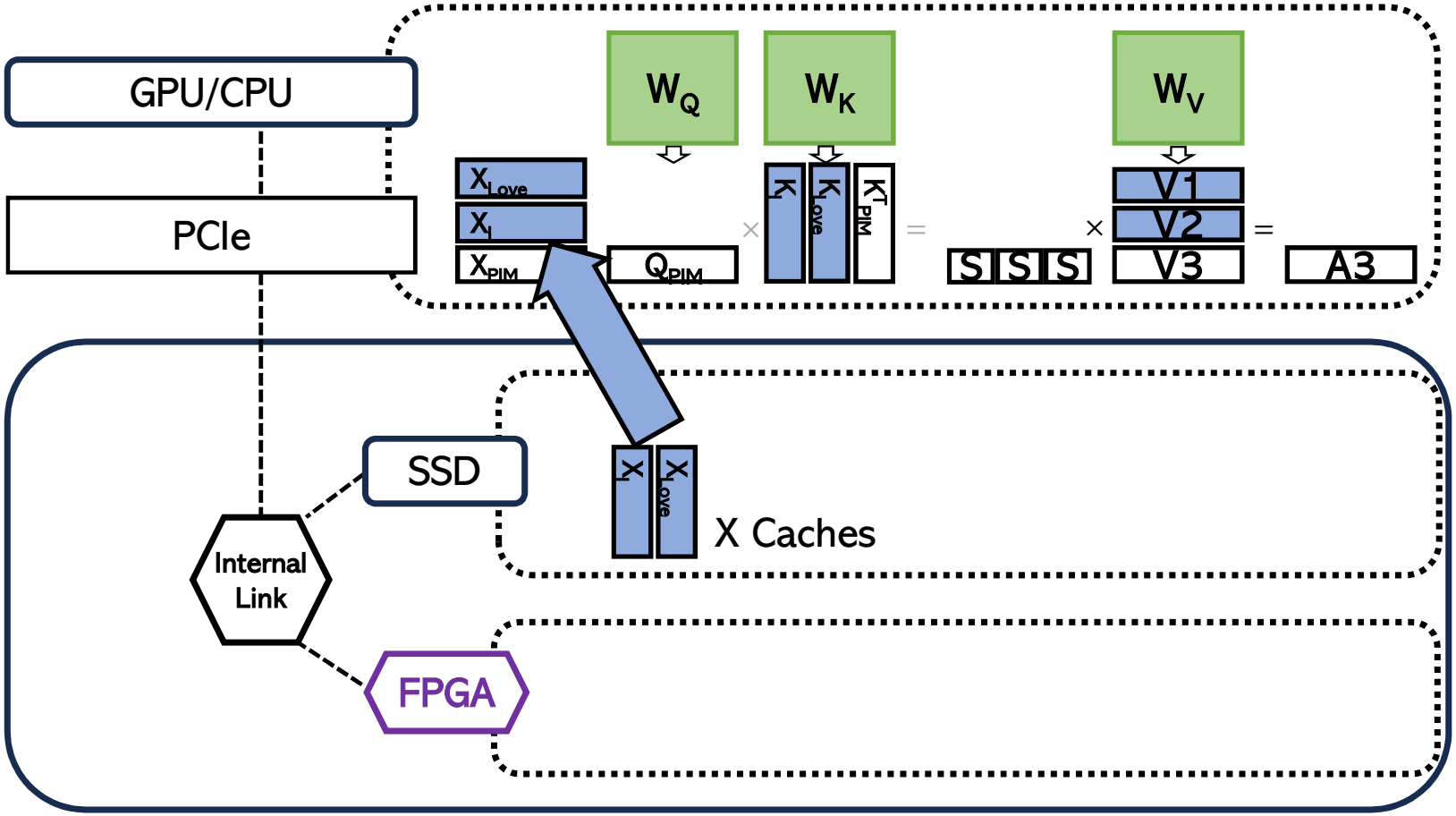
- Temporal KV buffer in CPU memory
- Flush when sufficient amount of KV is accumulated

Cooperative X-Cache (X)



- The new bottleneck is between the local SSD-FPGA
- Because the context is long…

Cooperative X-Cache (X)

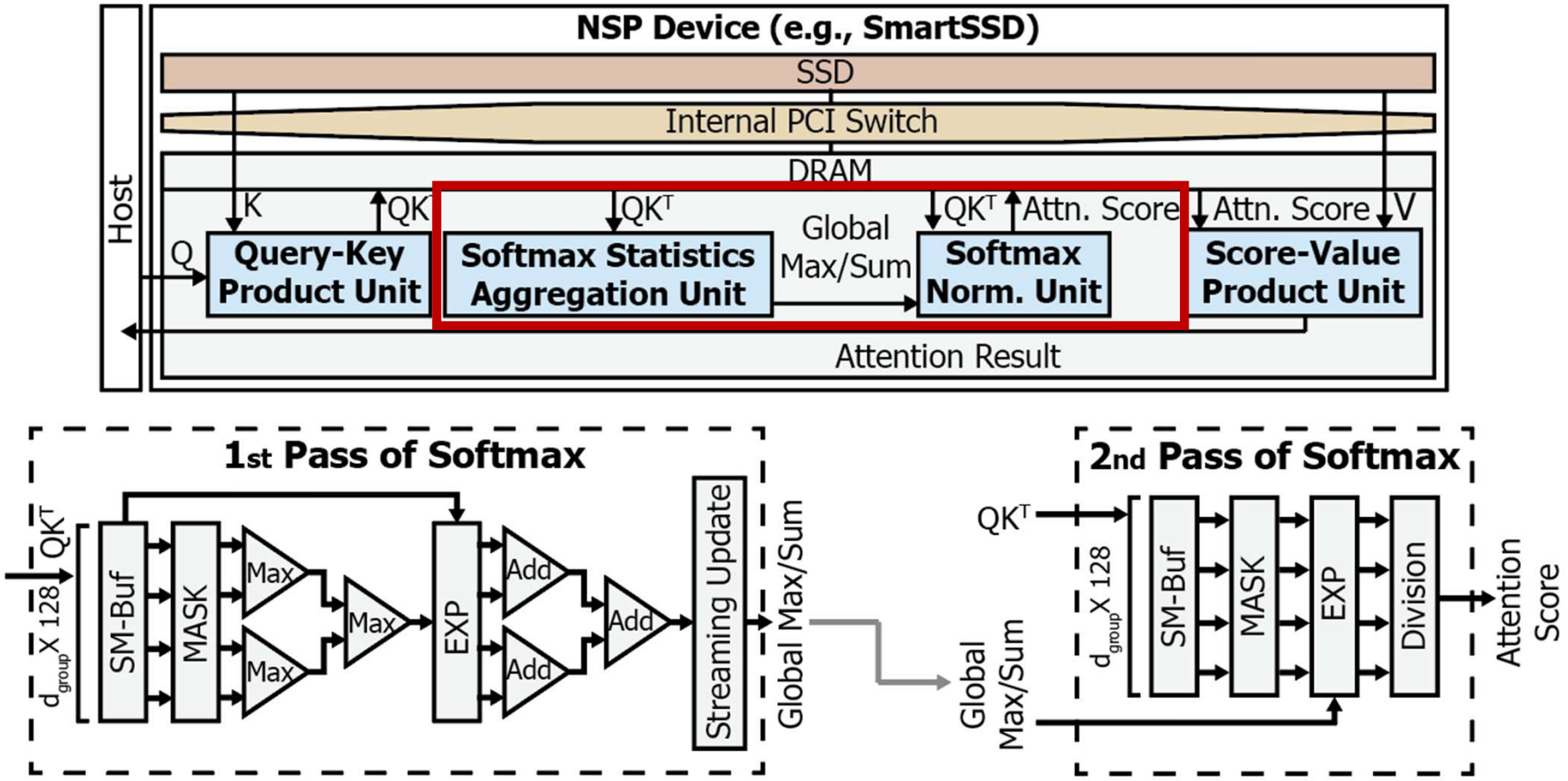


- Some attention heads are re-calculated from activation (X) in GPU
- Reduces SSD-side traffic to half, at the cost of GPU computation



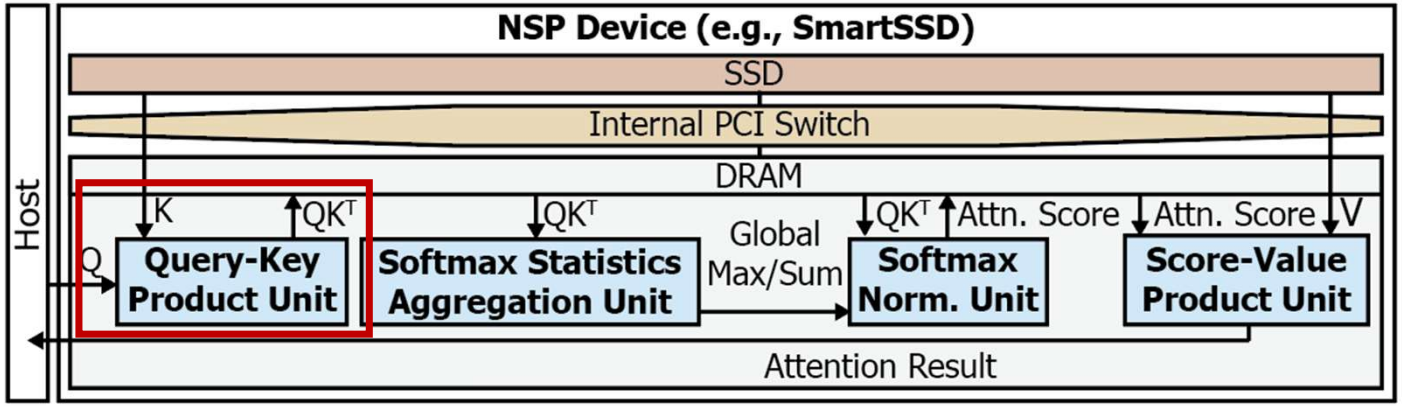
Architecture Details

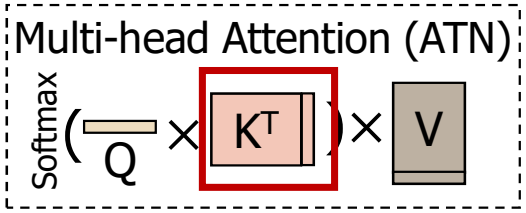
- ✓ Resource Efficient Two Pass Softmax
  to balance internal DRAM access and computational resource

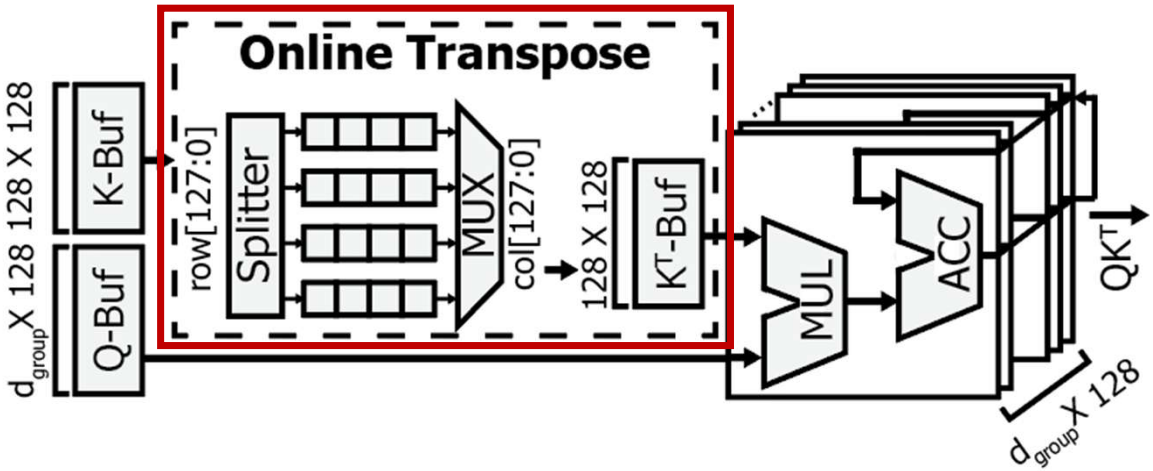


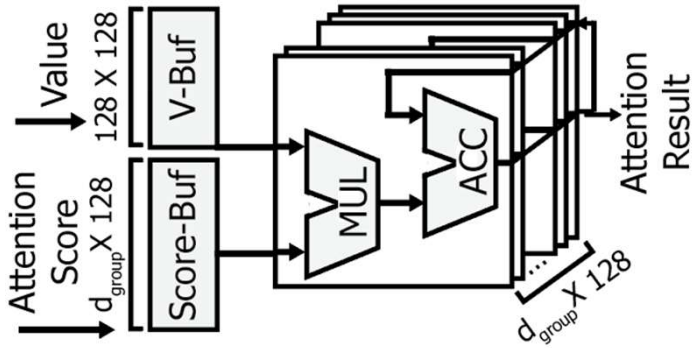
Architecture Details

- ✓ Online Transpose for storage layout (Key matrix needs to be transposed)



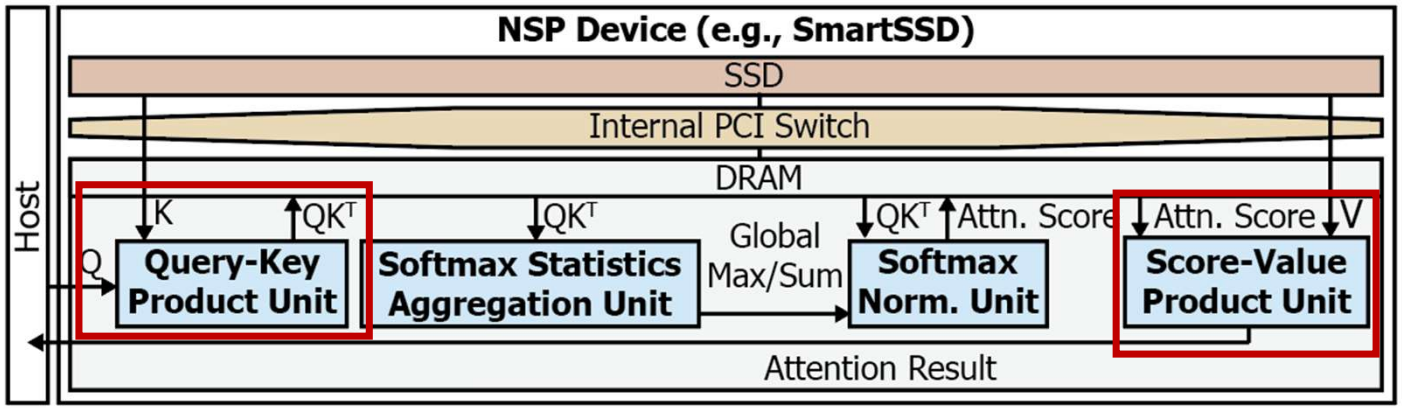


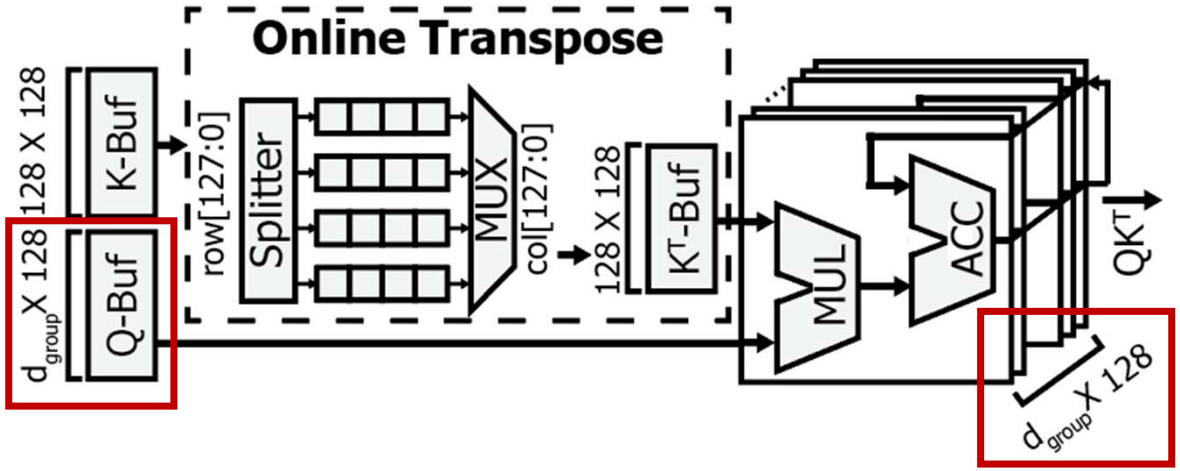


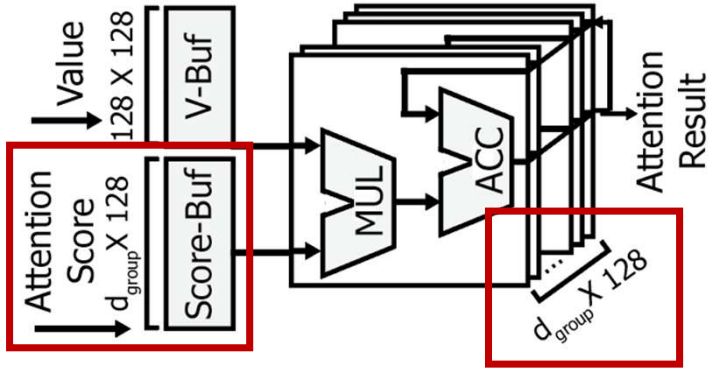


Architecture Details

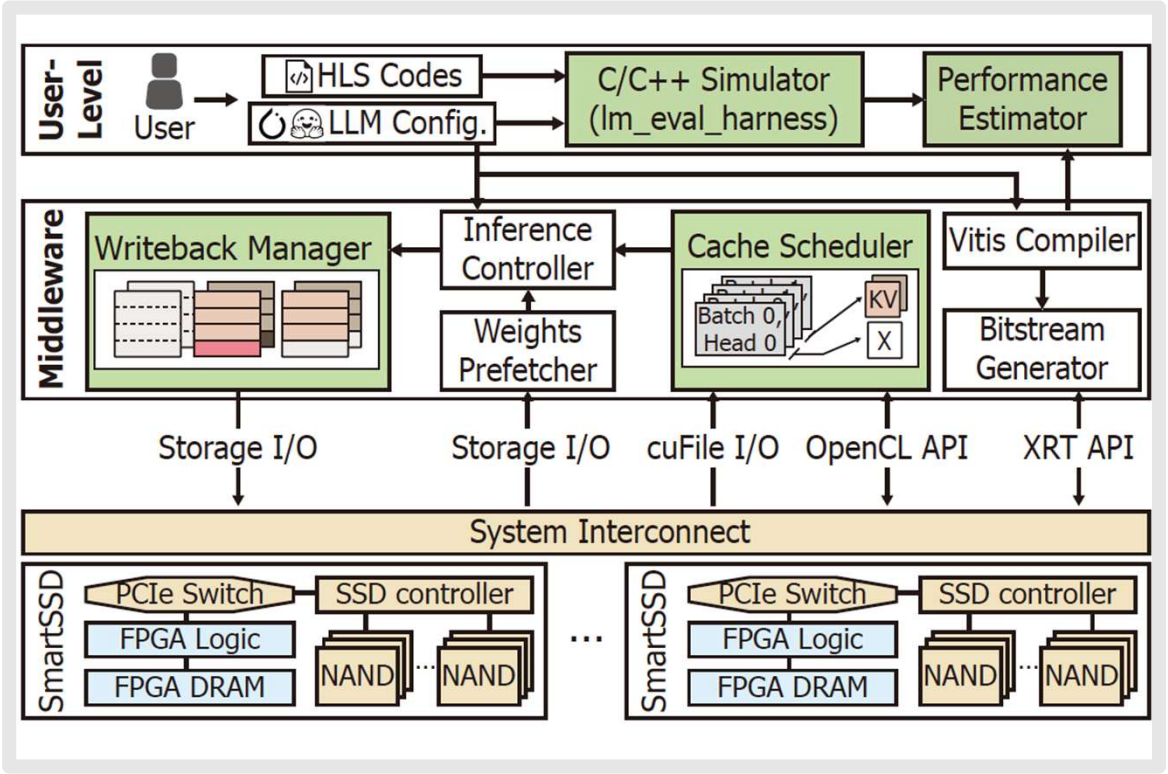
- ✓ Native Grouped Query Attention Support with flexible ALUs







Overall System & Experimental Setup



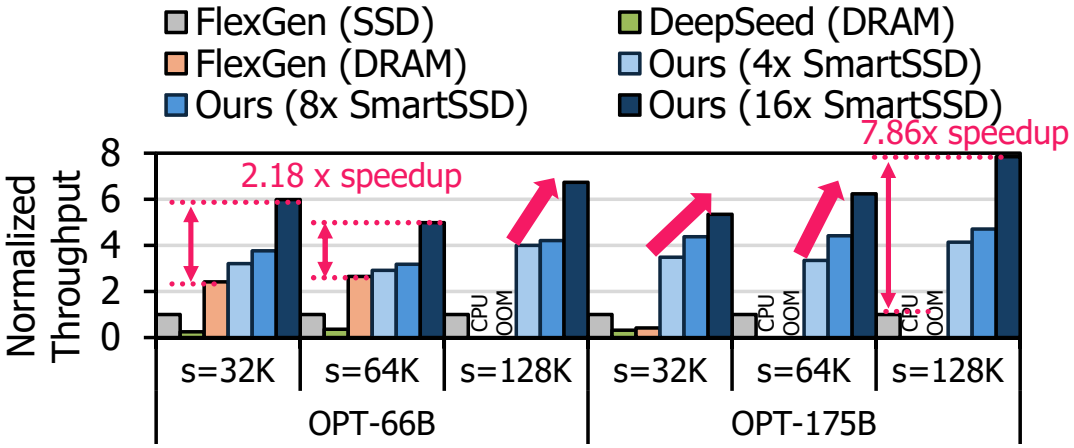
Experimental Setup



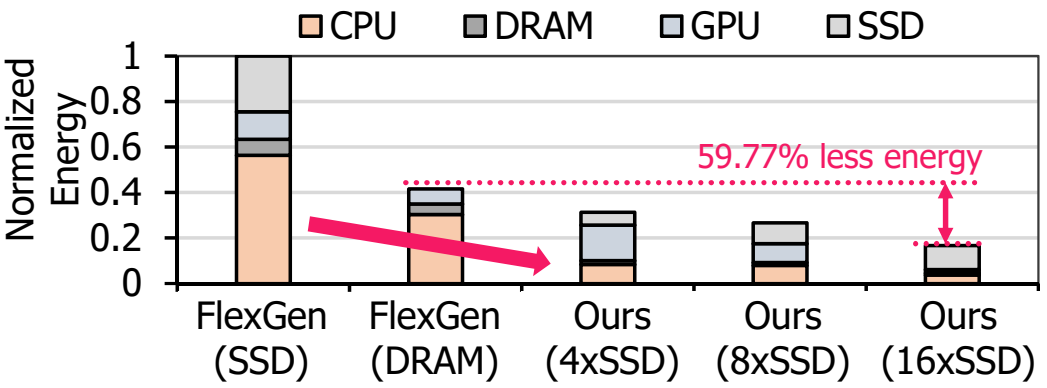
Baseline Method	KV Storage
FlexGen (SSD)	4x PCIe 4.0 SSD
FlexGen (DRAM)	16x32GB DDR4
DeepSpeed (DRAM)	16x32GB DDR4

- **SmartSSD:** Equipped with PCIe 3.0 SSD
- **GPU:** Single A100 (40GB) with FP16
- **Workload:** Batch sizes up to 16.

Experimental Results & Conclusion



#1) Performance Comparison

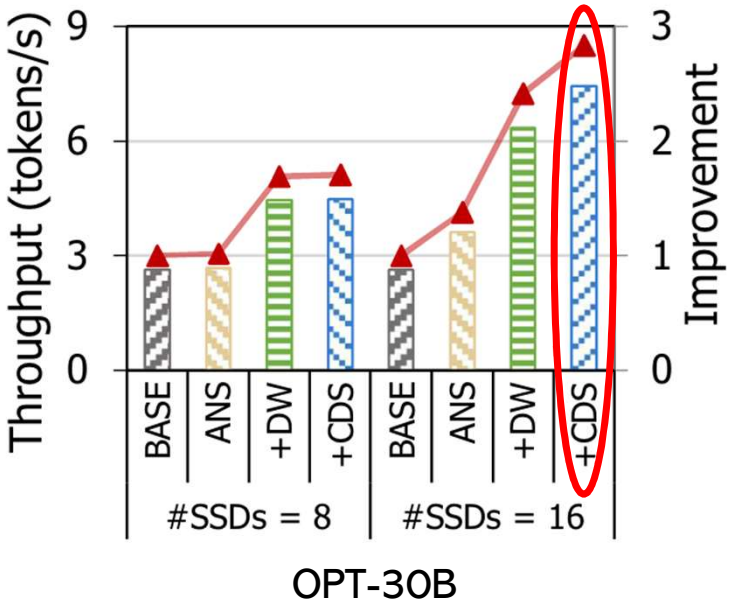


#2) Energy Breakdown Analysis

Key takeaways:

- ✓ **NSP:** A cost-effective solution for large-scale LLM offline inference.
- ✓ **Host-NSP cooperation:** Essential for maximizing system throughput.
- ✓ **Real-world implementation:** Uncovering bottlenecks hidden in simulations.

Performance Improvement



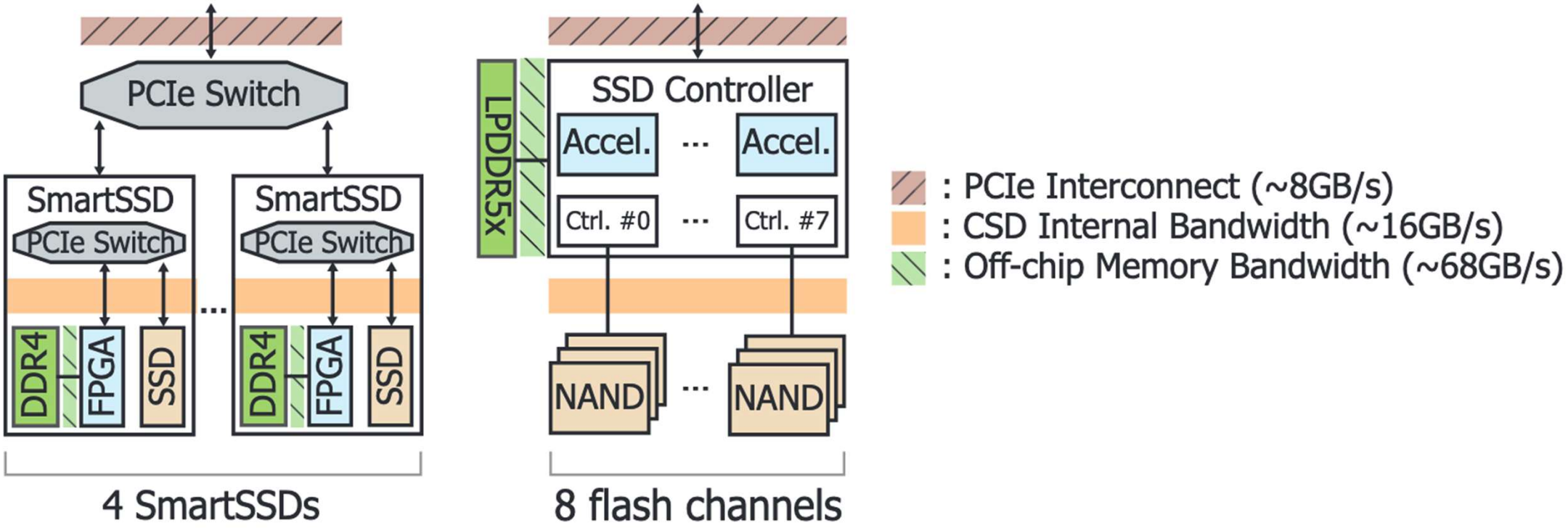
OPT-30B

- HILOS (ASPLOS 2026)
  - Is fully open-sourced, including the FPGA and SW
- ~2x speedup over the baseline
- More speedup for large models, longer context
- FPGA resource optimization was a bit difficult due to softmax



Compared to ISP's Sparse Attention

- ✓ 4x SmartSSD roughly matches 1x future NDP-SSD



Summary

- Training/Serving LLMs requires lots of computational power and memory
- Optimizing the data transfer overhead is the key
  - Configuring to use faster interconnects
  - Compressing communications
  - System-level designs
- All of these are open-sourced in Github including the FPGA design
  - Smart-Infinity (Training)
  - HILOS (Inference)
- leejinho@snu.ac.kr

